Preprocessing NLP Using NLTK Package

CSE 401: Artificial Intelligence

Utkarsh Gupta
A2305217557
7CSE 8Y

October 04th, 2020

1 Preprocessing

In this lab, we will be exploring how to preprocess tweets for sentiment analysis. We will provide a function for preprocessing tweets during this week's assignment, but it is still good to know what is going on under the hood. By the end of this lab, you will see how to use the NLTK package to perform a preprocessing pipeline for Twitter datasets.

1.1 Setup

In this lab, we will be using the Natural Language Toolkit (NLTK) package, an open-source Python library for natural language processing. It has modules for collecting, handling, and processing Twitter data.

For this exercise, we will use a Twitter dataset that comes with NLTK. This dataset has been manually annotated and serves to establish baselines for models quickly. Let us import them now as well as a few other libraries we will be using.

[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
```

1.2 About the Twitter dataset

The sample dataset from NLTK is separated into positive and negative tweets. It contains 5000 positive tweets and 5000 negative tweets exactly. The exact match between these classes is not a coincidence. The intention is to have a balanced dataset. That does not reflect the real distributions of positive and negative classes in live Twitter streams. It is just because balanced datasets simplify the design of most computational methods that are required for sentiment analysis. However, it is better to be aware that this balance of classes is artificial. In a local computer however, you can download the data by doing:

[2]:
```
# downloads sample twitter dataset. execute the line below if running on a local␣
 ↪machine.
nltk.download('twitter_samples')
```

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

[2]:
```
True
```

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

[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

[4]:
```
print('Number of positive tweets: ', len(all_positive_tweets))
print('Number of negative tweets: ', len(all_negative_tweets))

print('\nThe type of all_positive_tweets is: ', type(all_positive_tweets))
print('The type of a tweet entry is: ', type(all_negative_tweets[0]))
```

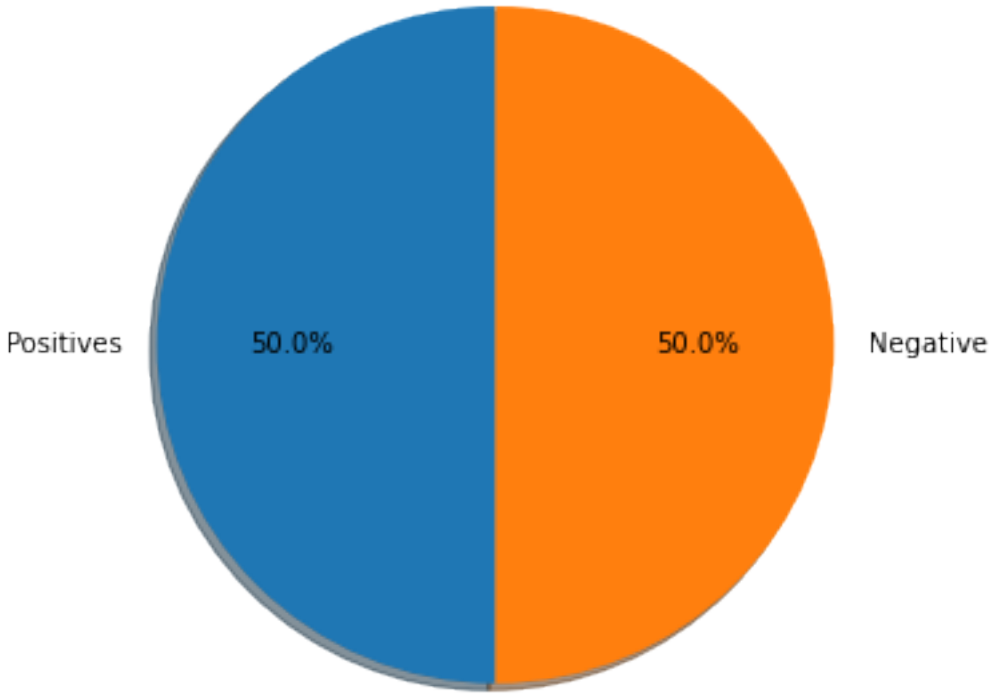
```
Number of positive tweets:  5000
Number of negative tweets:  5000

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

We can see that the data is stored in a list and as you might expect, individual tweets are stored as strings.

You can make a more visually appealing report by using Matplotlib's pyplot library. Let us see how to create a pie chart to show the same information as above. This simple snippet will serve you in future visualizations of this kind of data.

[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()
```



1.3 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.

[6]:
```
# 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[random.randint(0,5000)])

# print negative in red
print('\033[91m' + all_negative_tweets[random.randint(0,5000)])
```

```
i love airports :-):-))

I Blame Rantie For All Of This :(
```

One observation you may have is the presence of emoticons and URLs in many of the tweets. This info will come in handy in the next steps.

1.4 Preprocess raw text for Sentiment analysis

Data preprocessing is one of the critical steps in any machine learning project. It includes cleaning and formatting the data before feeding into a machine learning algorithm. For NLP, the preprocessing steps are comprised of the following tasks:

- Tokenizing the string
- Lowercasing
- Removing stop words and punctuation
- Stemming

The videos explained each of these steps and why they are important. Let's see how we can do these to a given tweet. We will choose just one and see how this is transformed by each preprocessing step.

[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
```

Let's import a few more libraries for this purpose.

[8]:
```
# download the stopwords from NLTK
nltk.download('stopwords')
```

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

[8]:
```
True
```

[9]:
```
#Import the necessary libraries
import re                                      # library for regular expression␣
 ↪operations
import string                                  # for string operations
```

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

1.4.1 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 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. `''`)

[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/3tfYomON1i


My beautiful sunflowers on a sunny Friday morning off :) sunflowers favourites

happy Friday off...
```

1.4.2 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 module from NLTK allows us to do these easily:

[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

favourites happy Friday off...


Tokenized string:

['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morning',

'off', ':)', 'sunflowers', 'favourites', 'happy', 'friday', 'off', '...']
```

1.4.3 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.

[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', 'yourselves', 'he',
'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's",
'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what',
'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is',
'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why',
```

```
'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some',
'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn',
"couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn',
"hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't",
'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
"shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
"wouldn't"]
```

```
Punctuation
```

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

We can see that the stop words list above contains some words that could be important in some contexts. These could be words like *i, not, between, because, won, against*. You might need to customize the stop words list for some applications. For our exercise, we will use the entire list.

For the punctuation, we saw earlier that certain groupings like ':)' and '...' should be retained when dealing with tweets because they are used to express emotions. In other contexts, like medical analysis, these should also be removed.

Time to clean up our tokenized tweet!

[13]:
```
print()
print('\033[92m')
print(tweet_tokens)
print('\033[94m')

tweets_clean = []

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
        tweets_clean.append(word)

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

```
['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morning',
'off', ':)', 'sunflowers', 'favourites', 'happy', 'friday', 'off', '...']


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

Please note that the words **happy** and **sunny** in this list are correctly spelled.

1.4.4 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 module which uses the Porter Stemming Algorithm. Let's see how we can use it in the cell below.

[14]:
```
print()
print('\033[92m')
print(tweets_clean)
print('\033[94m')

# Instantiate stemming class
stemmer = PorterStemmer()

tweets_stem = []

for word in tweets_clean:
    stem_word = stemmer.stem(word)  # stemming word
    tweets_stem.append(stem_word)  # append to the list

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

```
['beautiful', 'sunflowers', 'sunny', 'friday', 'morning', ':)', 'sunflowers',
'favourites', 'happy', 'friday', '...']


stemmed words:
['beauti', 'sunflow', 'sunni', 'friday', 'morn', ':)', 'sunflow', 'favourit',
'happi', 'friday', '...']
```

That's it! Now we have a set of words we can feed into to the next stage of our machine learning project.

1.5 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.

[15]:
```python
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
```

[16]:
```
# choose the same tweet
tweet = all_positive_tweets[2277]

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

# 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


preprocessed tweet:
['beauti', 'sunflow', 'sunni', 'friday', 'morn', ':)', 'sunflow', 'favourit',
'happi', 'friday', '...']
```

---

2 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.

[17]:
```
# Concatenate the lists, 1st part is the positive tweets followed by the negative
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
```

Next, we will build a labels array that matches the sentiments of our tweets. This data type works pretty much like a regular list but is optimized for computations and manipulation. The `labels` array will be composed of 10000 elements. The first 5000 will be filled with `1` labels denoting positive sentiments, and the next 5000 will be `0` labels denoting the opposite. We can do this easily with a series of operations provided by the `numpy` library:

- `np.ones()` - create an array of 1's
- `np.zeros()` - create an array of 0's
- `np.append()` - concatenate arrays

[18]:
```
# make a numpy array representing labels of the tweets
labels = np.append(np.ones((len(all_positive_tweets))), np.
 ↪zeros((len(all_negative_tweets))))
```

2.1 Dictionaries

In Python, a dictionary is a mutable and indexed collection. It stores items as key-value pairs and uses hash tables 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.

2.1.1 Definition

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

[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), and in this case, we used strings. We can also use floats, integers, tuples, etc.

2.1.2 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.

[20]:
```
# Add a new entry
dictionary['key3'] = -5

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

print(dictionary)
```

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

2.1.3 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() method: This allows us to set a default value if the dictionary key does not exist.

Let us see these in action:

[21]:
```
# Square bracket lookup when the key exist
print(dictionary['key2'])
```

```
2
```

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

[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:

2.2 This prints a value

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

2.3 Same as what you get with get

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

[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
```

2.4 Word frequency dictionary

Now that we know the building blocks, let's finally take a look at the **build_freqs()** function below. This is the function that creates the dictionary containing the word counts from each corpus.

[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.
    freqs = {}
```

```
    for y, tweet in zip(yslist, tweets):
        for word in process_tweet(tweet):
            pair = (word, y)
            if pair in freqs:
                freqs[pair] += 1
            else:
                freqs[pair] = 1

    return freqs
```

[25]:
```
# Call the build_freqs function to create frequency dictionary based on tweets␣
↪and labels
freqs = build_freqs(tweets, labels)

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

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

```
type(freqs) = <class 'dict'>
len(freqs) = 13067
```

[26]:
```
# prinf all the key-value pair of frequency dictionary
print(freqs)
```

```
{('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):
<snip>
('ban', 0.0): 1, ('failsatlif', 0.0): 1, ('press', 0.0): 1, ('duper', 0.0): 1,
('waaah', 0.0): 1, ('jaebum', 0.0): 1, ('ahmad', 0.0): 1, ('maslan', 0.0): 1,
('hull', 0.0): 1, ('misser', 0.0): 1}
```

Unfortunately, this does not help much to understand the data. It would be better to visualize this output to gain better insights.

---

3 Table of word counts

We will select a set of words that we would like to visualize. It is better to store this temporary information in a table that is very easy to use later.

[27]:
```
# select some words to appear in the report. we will assume that each word is␣
 ↪unique (i.e. no duplicates)
keys = ['happi', 'merri', 'nice', 'good', 'bad', 'sad', 'mad', 'best', 'pretti',
        '', ':)', ':(', '', '', '', '', '',
        'song', 'idea', 'power', 'play', 'magnific']



#each element consist of a sublist with this pattern: [<word>, <positive_count>,␣
 ↪<negative_count>].
data = []

# 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
```

[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],
```

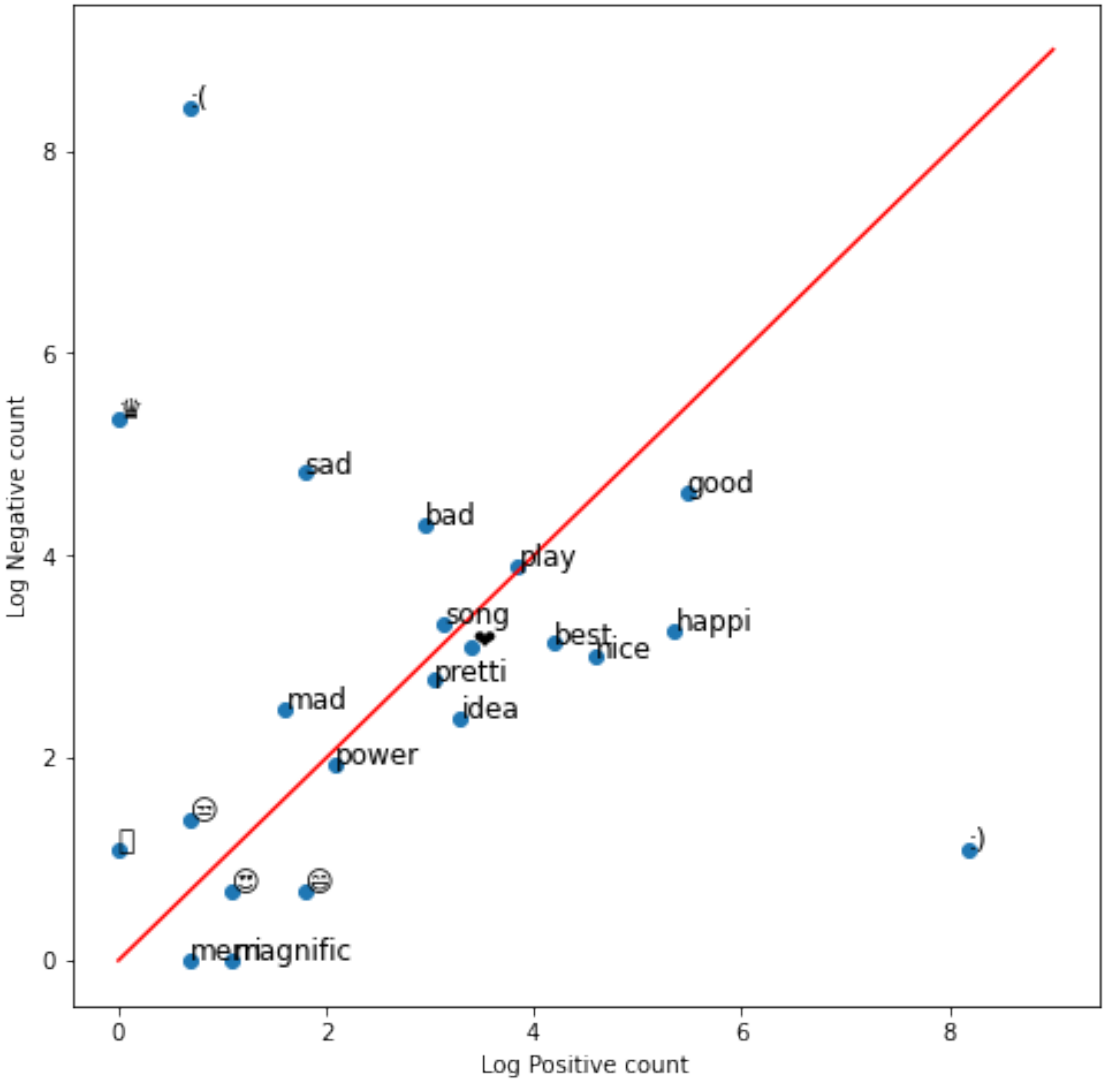
```
 [':)', 3568, 2],
 [':(', 1, 4571],
 ['', 1, 3],
 ['', 0, 2],
 ['', 5, 1],
 ['', 2, 1],
 ['', 0, 210],
 ['song', 22, 27],
 ['idea', 26, 10],
 ['power', 7, 6],
 ['play', 46, 48],
 ['magnific', 2, 0]]
```

We can then use a scatter plot to inspect this table visually. Instead of plotting the raw counts, we will plot it in the logarithmic scale to take into account the wide discrepancies between the raw counts (e.g. :) has 3568 counts in the positive while only 2 in the negative). The red line marks the boundary between positive and negative areas. Words close to the red line can be classified as neutral.

[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␣
 ↪before
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()
```

```
/usr/lib/python3/dist-packages/matplotlib/backends/backend_agg.py:238:
RuntimeWarning: Glyph 128556 missing from current font.
  font.set_text(s, 0.0, flags=flags)
/usr/lib/python3/dist-packages/matplotlib/backends/backend_agg.py:201:
RuntimeWarning: Glyph 128556 missing from current font.
```

```
font.set_text(s, 0, flags=flags)
```



This chart is straightforward to interpret. It shows that emoticons `:)` and `:(` are very important for sentiment analysis. Thus, we should not let preprocessing steps get rid of these symbols!

Furthermore, what is the meaning of the crown symbol? It seems to be very negative!

---

4 Create a features for the twitter dataset and Apply any machine learning algorithm to classify the tweets and check the accuracy of your model. Try with Logistic Regression for Classification

[29]:
```
import pandas as pd
data = []

for i in all_positive_tweets:
  data.append([i,1])
for i in all_negative_tweets:
   data.append([i,0])
```

[30]:
```
data = pd.DataFrame(data,columns = ['tweet','label'])
print(data.head())
```

```
                                               tweet  label
0  #FollowFriday @France_Inte @PKuchly57 @Milipol...      1
1  @Lamb2ja Hey James! How odd :/ Please call our...      1
2  @DespiteOfficial we had a listen last night :)...      1
3                                @97sides CONGRATS :)      1
4  yeaaaah yippppy!!!  my accnt verified rqst has...      1
```

[31]:
```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
x = data['tweet']
y = data['label']
```

[32]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state␣
↪= 53)
```

[34]:
```
count_vectorizer = CountVectorizer(stop_words='english',min_df=6)
count_train = count_vectorizer.fit_transform(x_train)
count_test = count_vectorizer.transform(x_test)
print(count_vectorizer.get_feature_names()[:10])
print(count_vectorizer.vocabulary_.keys())
```

```
['000', '07', '10', '100', '11', '12', '13', '15', '20', '2015']
dict_keys(['happy', 'friday', 'just', 'enjoyed', 'super', '100', 'pleasure',
'let', 'know', 'need', 'http', 'nice', 'day', 'queen', 'hi', 'ask', 'fingers',
'cut', 'food', 'shop', 'week', 'https', '5sos', 'shot', 'listening', 'days',
'think', 'love', 'meet', 'person', 'guess', 'spent', 'half', 'outside', 'bc',
'left', 'work', 'hope', 'needs', 'leave', 'soon', 'party', 'cheers', 'cause',
'haha', 'maybe', 'thanks', 'rest', 'sounds', 'lovely', 'don', 'plans', 'add',
'new', 'moment', 'inside', 'today', 'asked', 'members', 'favourite', 'brain',
'thought', 'friend', 'lot', 'makes', 'followed', 'justinbieber', 'agree',
'phone', 'saturday', 'fuck', 'ill', 'promise', 'dog', 'stress', 'haven', 'seen',
'years', 'oh', 'sorry', 'hear', 'having', 'tuesday', 'want', 'finally', 'bed',
```

'good', 'night', 'hurt', 'allah', 'yes', 'better', 'plan', 'look', 'forward',
'school', 'weeks', 'feeling', 'saw', 'follow', 'like', 'dark', 'plz', 'far',
'fan', 'wanna', 'free', 'check', 'thx', 'drive', 'stats', 'arrived', 'follower',
'unfollowers', 'going', 'spend', 'leaving', 'tomorrow', 'ok', 'chill', 'aw',
'babe', 'england', 'll', 'turn', 'welcome', 'amp', 'sharing', 'enjoy', 'app',
'wishing', 'weekend', 'ya', 'green', 'tea', 'bored', 'did', 'crying', 'red',
'train', 'retweet', 'active', 'followers', 'getting', 'sick', 'late', 'event',
'sure', 'stuff', 'august', 'thank', 'sleep', 'sweet', 'dreams',
'zayniscomingbackonjuly26', 'im', 'zayn', 'come', 'believe', 'omg', 'birthday',
'missed', 'fun', 'home', 'face', 'couldn', 'kid', 'right', 'literally',
'people', 'actually', 'doing', 'thing', 'rock', 'tired', 'head', 'afternoon',
'lets', 'read', 'al', 'finish', 'man', 'long', 'does', 'wish', 'year', 've',
'watching', 'gutted', 'miss', 'lady', 'talk', 'help', 'watch',
'bajrangibhaijaanhighestweek1', 'wait', 'really', 'told', 'ran', 'away',
'forever', 'beat', 'visit', 'na', 'aren', 'say', 'proud', 'heart', 'broken',
'works', 'smile', 'mind', 'following', 'body', 'wonderful', 'way', 'guys',
'video', 'amazing', 'ohh', 'team', 'thats', 'question', 'happened', 'hey',
'text', 'july', '24', '2015', '07', 'yeah', 'fucking', 'email', 'information',
'pro', 'instead', 'set', 'isn', 'black', 'cat', 'bad', 'luck', 'kinda', 'ago',
'got', 'sent', 'songs', 'gt', 'stefaniescott', 'real', 'favorite', 'sun',
'beli', 'eve', 'wi', 'justi', 'x15', '', '', 'funny', 'dress', 'pretty',
'change', 'close', 'xd', 'feel', 'shit', 'morning', 'big', 'cool', 'dream',
'fab', 'release', 'win', 'huge', 'giveaway', 'running', 'sore', 'throat',
'fever', 'create', 'fucked', 'using', 'appreciate', 'support', 'vote', 'games',
'try', 'best', 'making', 'sad', 'glad', 'liked', 'sponsor', 'feels', 'shift',
'chocolate', 'xx', 'broke', 'money', 'dm', 'make', 'fast', 'received', 'wrong',
'order', 'sort', 'uk', 'okay', 'la', 'wont', 'thinking', 'ur', 'hate', 'shame',
'tweets', 'waiting', 'invite', 'working', 'hahaha', 'forgotten', 'sooo',
'jealous', 'heard', 'eat', 'weird', 'fine', 'game', 'success', 'ang', 'cold',
'store', 'time', 'doesn', 'update', 'mention', 'longer', 'awful', 'story',
'lost', 'id', 'airport', 'concert', 'god', 'damn', 'gonna', 'didnt', 'congrats',
'infinite', 'fav', 'song', 'rip', 'looks', 'awesome', 'cars', 'house', 'yay',
'isnt', 'loved', 'friends', 'rn', 'thankyou', 'sa', 'sir', 'group', 'facebook',
'small', 'lt', 'harry', 'pls', 'bae', 'ignore', 'taking', 'reply', 'answer',
'pizza', 'wanted', '40', 'goodmorning', 'life', 'easy', 'account',
'bhaktisbanter', 'flipkartfashionfriday', 'worst', 'pain', 'bam',
'barsandmelody', 'bestfriend', '969horan696', 'loves', 'warsaw', 'hello',
'youth', 'job', 'opportunities', 'tolajobjobs', 'channel', 'worse', 'awake',
'yep', 'weather', 'bag', 'idk', 'play', 'talking', 'tweet', 'dont', 'hard',
'info', 'dying', 'little', 'coming', 'ff', 'old', 'pic', 'used', 'buy', 'meant',
'rude', 'mean', 'usually', 'use', 'dude', 'waste', 'twitter', 'tbh', 'blue',
'fb', '15', 'later', '10', 'snapchat', 'kik', 'chat', 'xxx', 'kikmeboys',
'travel', 'box', 'bring', 'beautiful', 'place', 'picture', 'won', 'able', '50',
'dead', '300', 'send', 'john', 'checked', 'tell', '20', 'mins', 'fair', 'cute',
'saying', 'short', 'finished', 'spain', 'excited', 'season', 'jnlazts',
'rcvcyyo0iq', 'youtube', 'link', 'trying', 'live', 'srsly', 'seeing',
'brilliant', 'gorgeous', 'shout', 'girl', 'came', 'went', 'college', 'baby',
'stop', 'playing', 'feelings', 'aint', 'hoping', 'tgif', 'tonight', 'view',

'eyes', 'notice', 'great', 'start', 'light', 'business', 'jaymcguiness',
'family', 'trip', 'early', 'paper', 'sigh', 'likeforlike', 'write', 'possible',
'months', '2nd', 'choice', 'iphone', 'facetime', 'pictures', 'cuz', 'sucks',
'badly', 'second', 'smiling', 'listen', 'fback', 'hugs', 'didn', 'definitely',
'took', 'followfriday', 'supports', 'community', 'photo', 'happens', 'kidding',
'littlemix', 'girls', 'says', 'available', 'awwww', 'forget', 'music',
'looking', 'kikgirl', 'french', 'model', 'watched', 'wow', 'scared', 'af',
'course', 'waking', 'catch', 'barely', 'dinner', 'date', 'luke', 'reason',
'youre', 'whats', 'ice', 'cream', 'missing', 'needed', 'uniteblue', 'tcot',
'sadly', 'remember', 'lol', 'hug', 'class', 'women', 'art', 'everyday',
'breakfast', 'zaynmalik', 'followback', 'meeting', 'gets', 'amber', 'post',
'flight', 'tried', 'btw', 'quite', 'wasn', 'men', 'tour', 'said', 'board',
'bye', 'sound', 'decide', 'book', 'hot', 'kiksex', 'tagsforlikes', 'reading',
'interesting', 'kind', 'totally', 'stay', 'impastel', 'city', 'country', 'ones',
'met', 'stuck', 'certain', 'case', 'message', 'easier', 'craving', 'lunch',
'started', 'things', 'busy', 'truth', 'die', 'save', 'bday', 'boring',
'original', 'door', 'guy', 'poor', 'english', 'worry', 'bit', 'hair', 'pass',
'words', 'past', 'news', 'wforwoman', 'wsalelove', 'sign', 'wouldn', 'end',
'yup', '30', 'lucky', 'nope', 'air', 'line', 'care', 'ha', 'touch', 'forgot',
'gold', 'stream', 'keeps', 'bath', 'goodnight', 'park', 'hotel', 'hopefully',
'hours', 'ahh', 'ate', 'unfortunately', 'calm', 'tummy', 'hurts', 'bro',
'coffee', 'special', 'address', 'problem', 'power', 'festival', 'ive', 'times',
'world', 'gift', 'true', 'bank', 'cake', 'car', 'regret', 'probably', 'fix',
'plays', 'windows', 'cheese', 'white', 'near', 'exactly', 'list', 'ty', 'mum',
'london', 'lonely', 'issue', 'future', 'starts', 'cutie', 'congratulations',
'hornykik', 'internet', 'means', 'aww', 'stomach', 'number', 'woke', 'shopping',
'parents', 'vacation', 'ah', 'anybody', 'added', 'final', 'design',
'ext098yq1b', 'books', 'hun', 'ball', 'questions', 'episode', 'worth', 'learn',
'lots', 'hehe', 'high', 'kids', 'wonder', 'boy', 'wants', 'rain', 'breaking',
'chris', 'selfie', 'alice', 'pics', 'emilybett', 'teenchoice',
'choiceinternationalartist', 'superjunior', 'bet', 'site', '000', 'pool',
'awww', 'album', 'ready', 'goes', 'da', 'wife', 'kikmenow', 'summer', 'likes',
'open', 'walk', 'hungry', 'ain', 'online', 'feedback', 'perfect', 'dots',
'braindots', 'hour', 'rafaelallmark', 'supporting', 'blog', 'interested',
'lmao', 'month', 'website', 'follback', 'tho', 'ass', 'fall', 'asleep',
'tweeting', 'uberuk', 'uber', 'anymore', 'acc', 'load', 'fantastic', 'sending',
'tl', 'chance', 'wake', '11', 'sunshine', 'deserve', 'wtf', 'driving', 'ugly',
'dear', 'towns', 'movies', 'tv', 'skype', 'shows', 'painful', 'training', 'hai',
'gotta', 'join', 'yesterday', 'gone', 'absolutely', 'ugh', 'ubericecream',
'click', 'aqui', 'film', 'cover', 'felt', 'louis', 'influencers', 'young',
'bravefrontiergl', 'hurry', 'mad', 'bb', 'ears', 'surprise', 'selenagomez',
'bitch', 'crazy', 'kikhorny', 'happiness', 'appreciated', 'havent', 'earlier',
'videos', 'share', 'boys', 'idea', 'low', 'moving', 'bby', 'especially',
'turned', 'middle', 'lil', 'giving', 'offer', 'oppa', 'single', 'download',
'lose', 'liam', 'card', 'wet', 'indiemusic', 'sexy', 'failed', 'till',
'mistake', 'understand', 'paid', 'pay', 'rt', 'tom', 'happen', 'different',
'voice', 'apparently', 'fans', 'mom', 'hit', 'run', 'contact', 'movie',
'adeccowaytowork', 'kunoriforceo', 'ceo1month', 'sister', 'vidcon', 'hahahaha',

```
'ended', 'shall', 'review', 'raining', 'beach', 'holiday', 'outfit', '13',
'fact', 'hold', 'kikme', 'real_liam_payne', 'fell', 'knows', 'sex', 'gave',
'alright', 'point', 'tickets', 'hell', 'asking', 'joined', 'keeping', 'office',
'count', 'points', 'collection', 'dad', 'computer', 'conversation', 'station',
'leeds', 'ye', 'se', 'ticket', 'garden', 'solo', 'match', 'mr', 'pick', 'area',
'monday', 'water', 'ko', 'sunday', 'choose', 'details', 'goodbye', 'matt',
'warm', 'couple', 'slept', 'mood', 'pa', 'seriously', 'regular', 'quick',
'comes', 'eating', 'sleeping', 'safe', 'stage', 'realized', 'unless',
'planning', 'completely', 'madrid', 'kit', 'instagram', 'stupid', 'huhu',
'secret', 'pre', 'schedule', 'page', 'photos', '12', 'horrible', 'mm', 'drop',
'writing', 'kiksexting', 'annoying', 'cc', 'article', 'carterreynolds', '33',
'entire', 'word', 'clothes', 'australia', 'til', 'looked', 'got7', 'brother',
'luv'])
```

[35]:
```
print(count_vectorizer.get_params(deep=True))
```

```
{'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype': <class
'numpy.int64'>, 'encoding': 'utf-8', 'input': 'content', 'lowercase': True,
'max_df': 1.0, 'max_features': None, 'min_df': 6, 'ngram_range': (1, 1),
'preprocessor': None, 'stop_words': 'english', 'strip_accents': None,
'token_pattern': '(?u)\\b\\w\\w+\\b', 'tokenizer': None, 'vocabulary': None}
```

[36]:
```
count_df = pd.DataFrame(count_train.A, columns=count_vectorizer.
 ↪get_feature_names())
print(count_df)
```

```
      000  07  10  100  11  12  13  15  20  2015  ...  young  youre  youth  \
0       0   0   0    0   0   0   0   0   0     0  ...      0      0      0
1       0   0   0    1   0   0   0   0   0     0  ...      0      0      0
2       0   0   0    0   0   0   0   0   0     0  ...      0      0      0
3       0   0   0    0   0   0   0   0   0     0  ...      0      0      0
4       0   0   0    0   0   0   0   0   0     0  ...      0      0      0
...   ...  ..  ..  ...  ..  ..  ..  ..  ..   ...  ...    ...    ...    ...
6995    0   0   0    0   0   0   0   0   0     0  ...      0      0      0
6996    0   0   0    0   0   0   0   0   0     0  ...      0      0      0
6997    0   0   0    0   0   0   0   0   0     0  ...      0      0      0
6998    0   0   0    0   0   0   0   0   0     0  ...      0      0      0
6999    0   0   0    0   0   0   0   0   0     0  ...      0      0      0

      youtube  yup  zayn  zayniscomingbackonjuly26  zaynmalik
0           0    0     0                         0          0   0    0
1           0    0     0                         0          0   0    0
2           0    0     0                         0          0   0    0
3           0    0     0                         0          0   0    0
4           0    0     0                         0          0   0    0
...       ...  ...   ...                       ...        ...  ..  ...
6995        0    0     0                         0          0   0    0
6996        0    0     0                         0          0   0    0
```

```
6997        0    0     0                          0          0    0     0
6998        0    0     0                          0          0    0     0
6999        0    0     0                          0          0    0     0

[7000 rows x 959 columns]
```

[37]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

clf = MultinomialNB()
clf.fit(count_train,y_train)
pred = clf.predict(count_test)
```

[38]:
```
score = metrics.accuracy_score(y_test,pred)
print(score)

cm = metrics.confusion_matrix(y_test,pred,labels = [1,0])
print(cm)
```

```
0.742
[[1069  466]
 [ 308 1157]]
```

[39]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

clf = SVC()
```

[40]:
```
clf.fit(count_train,y_train)
pred = clf.predict(count_test)
```

[41]:
```
score = metrics.accuracy_score(y_test,pred)
print(score)

cm = metrics.confusion_matrix(y_test,pred,labels = [1,0])
print(cm)
```

```
0.7383333333333333
[[1030  505]
 [ 280 1185]]
```