NLP_C1_W1_lecture_nb_02-Copy1

December 20, 2020

1 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.

1.1 Setup

Let's import the required libraries for this lab:

```
In [1]: import nltk                                # Python library for NLP
        from nltk.corpus import twitter_samples    # sample Twitter dataset from NLTK
        import matplotlib.pyplot as plt            # visualization library
        import numpy as np                         # library for scientific computing and ma
```

Import some helper functions that we provided in the utils.py file:

- `process_tweet()`: Cleans the text, tokenizes it into separate words, removes stopwords, and converts words to stems.
- `build_freqs()`: This counts how often a word in the 'corpus' (the entire set of tweets) was associated with a positive label `1` or a negative label `0`. It then builds the `freqs` dictionary, where each key is a `(word,label)` tuple, and the value is the count of its frequency within the corpus of tweets.

```
In [2]: # download the stopwords for the process_tweet function
        nltk.download('stopwords')

        # import our convenience functions
        from utils import process_tweet, build_freqs
```

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

1.2 Load the NLTK sample dataset

As in the previous lab, we will be using the Twitter dataset from NLTK.

```
In [5]: # select the lists of positive and negative tweets
        all_positive_tweets = twitter_samples.strings('positive_tweets.json')
        all_negative_tweets = twitter_samples.strings('negative_tweets.json')

        # 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

```
In [6]: # make a numpy array representing labels of the tweets
        labels = np.append(np.ones((len(all_positive_tweets))), np.zeros((len(all_negative_twe
```

1.3 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.

1.3.1 Definition

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

```
In [8]: 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.

1.3.2 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.

```
In [9]: # Add a new entry
        dictionary['key3'] = -5

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

        print(dictionary)
```

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

1.3.3 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:

```
In [10]: # Square bracket lookup when the key exist
         print(dictionary['key2'])
```

```
2
```

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

```
In [11]: # The output of this line is intended to produce a KeyError
         print(dictionary['key8'])
```

```
        ---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call last)

        <ipython-input-11-8d63520997fb> 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:

```
In [12]: # 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))
```

```
item found:  0
item found:  0
```

```
In [13]: # 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
```

1.4 Word frequency dictionary

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

```
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
```

You can also do the for loop like this to make it a bit more compact:

```
for y, tweet in zip(yslist, tweets):
    for word in process_tweet(tweet):
        pair = (word, y)
        freqs[pair] = freqs.get(pair, 0) + 1
```

As shown above, each key is a 2-element tuple containing a `(word, y)` pair. The `word` is an element in a processed tweet while `y` is an integer representing the corpus: `1` for the positive tweets and `0` for the negative tweets. The value associated with this key is the number of times that word appears in the specified corpus. For example:

```
# "folowfriday" appears 25 times in the positive tweets
('followfriday', 1.0): 25

# "shame" appears 19 times in the negative tweets
('shame', 0.0): 19
```

Now, it is time to use the dictionary returned by the `build_freqs()` function. First, let us feed our `tweets` and `labels` lists then print a basic report:

```
In [14]: # create frequency dictionary
         freqs = build_freqs(tweets, labels)

         # check data type
         print(f'type(freqs) = {type(freqs)}')

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

```
type(freqs) = <class 'dict'>
len(freqs) = 13076
```

Now print the frequency of each word depending on its class.

```
In [ ]: print(freqs)
```

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

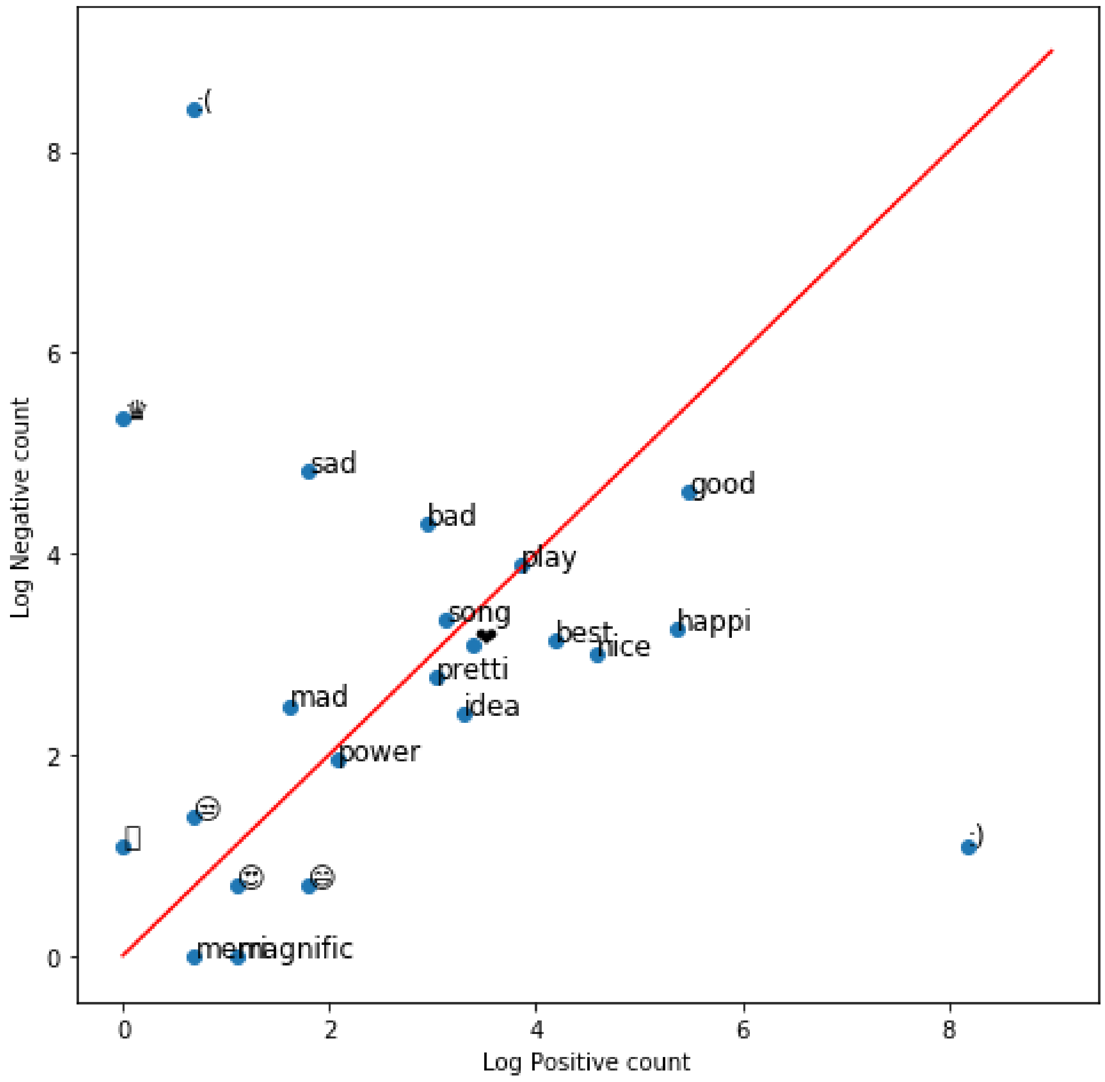
1.5 Table of word counts

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.

```
In [17]: # select some words to appear in the report. we will assume that each word is unique
         keys = ['happi', 'merri', 'nice', 'good', 'bad', 'sad', 'mad', 'best', 'pretti',
                 '', ':)', ':(', '', '', '', '', '',
                 'song', 'idea', 'power', 'play', 'magnific']

         # list representing our table of word counts.
         # each element consist of a sublist with this pattern: [<word>, <positive_count>, <ne
         data = []

         # loop through our selected words
         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[17]: [['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.

```
In [18]: 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()
```



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!

1.5.1 That's all for this lab! We've seen how to build a word frequency dictionary and this will come in handy when extracting the features of a list of tweets. Next up, we will be reviewing Logistic Regression. Keep it up!