NLP_C1_W1_lecture_nb_01

September 15, 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 lecture, you will see how to use the NLTK package to perform a preprocessing pipeline for Twitter datasets.

1.1 Setup

You will be doing sentiment analysis on tweets in the first two weeks of this course. To help with that, 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, and you will be acquainted with them as we move along the course.

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.

```
In [ ]: 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
```

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.

The dataset is already downloaded in the Coursera workspace. In a local computer however, you can download the data by doing:

```
In [ ]: # downloads sample twitter dataset. uncomment the line below if running on a local mac
        # nltk.download('twitter_samples')
```

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

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

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.

```
In [ ]: # 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. (Warning: This is taken from a public dataset of real tweets and a very small portion has explicit content.)

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

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.

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

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

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

```
In [ ]: 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. `''`)

```
In [ ]: 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)
```

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:

```
In [ ]: 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)
```

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.

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

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!

```
In [ ]: 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)
```

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.

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

        # Instantiate stemming class
        stemmer = PorterStemmer()
```

```
# Create an empty list to store the stems
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)
```

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. In the week's assignment, you will use the function `process_tweet(tweet)` available in *utils.py*. We encourage you to open the file and you'll see that this function's implementation is very similar to the steps above.

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 [ ]: from utils import process_tweet # Import the process_tweet function

        # choose the same tweet
        tweet = all_positive_tweets[2277]

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

        # call the imported function
        tweets_stem = process_tweet(tweet); # Preprocess a given tweet

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

That's it for this lab! You now know what is going on when you call the preprocessing helper function in this week's assignment. Hopefully, this exercise has also given you some insights on how to tweak this for other types of text datasets.