0101_tokenization_stemming_and_stop_word_removal

December 15, 2018

1 Pre-processing data: tokenization, stemming, and removal of stop words

Here we will look at three common pre-processing step sin natural language processing:

1) Tokenization: the process of segmenting text into words, clauses or sentences (here we will separate out words and remove punctuation).

2) Stemming: reducing related words to a common stem.

3) Removal of stop words: removal of commonly used words unlikely to be useful for learning.

We will load up 50,000 examples from the movie review database, imdb, and use the NLTK library for text pre-processing. The NLTK library comes with a standard Anaconda Python installation (www.anaconda.com), but we will need to use it to install the 'stopwords' corpus of words.

1.1 Downloading the NLTK library

This command will open the NLTK downloader. You may download everything from the collections tab. Otherwise, for this example you may just download 'stopwords' from the 'Corpora' tab.

```
In [1]: import nltk
        # To open dialog download:
        # nltk.download();

        # To downlaod just stopwords:
        nltk.download('stopwords');
```

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

1.2 Load data

If you have not previously loaded and saved the imdb data, run the following which will load the file from the internet and save it locally to the same location this is code is run from.

We will load data into a pandas DataFrame.

```
In [ ]: import pandas as pd

        file_location = 'https://gitlab.com/michaelallen1966/00_python_snippets' +\
            '_and_recipes/raw/master/machine_learning/data/IMDb.csv'
        imdb = pd.read_csv(file_location)
        # save to current directory
        imdb.to_csv('imdb.csv', index=False)
```

If you have already saved the data locally, load it up into memory:

```
In [2]: import pandas as pd

        imdb = pd.read_csv('imdb.csv')
```

Let's look at what columns exist in the imdb data:

```
In [3]: print (list(imdb))

['review', 'sentiment']
```

We'll pull out the first review and sentiment to look at the contents. The review is text and the sentiment label is either 0 (negative) or 1 (positive) based on how the reviewer rated it on imdb. We will use the Pnadas DataFrame .iloc method to selc the first review (which has an index of zero)

```
In [4]: print(imdb['review'].iloc[0])

I have no read the novel on which "The Kite Runner" is based. My wife and daughter, who did, t
```

```
In [5]: print(imdb['sentiment'].iloc[0])

1
```

1.3 Convert to lower case

We don't want words being trested differently dut to differences in upper and lower case, so we'll ocnvert all reviews to lower case, an dlook at the first review again.

```
In [6]: imdb['review'] = imdb['review'].str.lower()
        print(imdb['review'].iloc[0])

i have no read the novel on which "the kite runner" is based. my wife and daughter, who did, t
```

1.4 Tokenization

We will use word_tokenize method from NLTK to split the review text into individual words (and you will see that punctuation is also produced as separate 'words'). Let's look at our example row.

```
In [7]: import nltk
        print (nltk.word_tokenize(imdb['review'].iloc[0]))
```

```
['i', 'have', 'no', 'read', 'the', 'novel', 'on', 'which', '``', 'the', 'kite', 'runner', "''"
```

We will now apply the word_tokenize to all records, making a new column in our imdb DataFrame. Each entry will be a list of words. Here we will also strip out non alphanumeric words/characters (such as numbers and punctuation) using .isalpha (you could use .isalnum if you wanted to keep in numbers as well).

```
In [8]: def identify_tokens(row):
            review = row['review']
            tokens = nltk.word_tokenize(review)
            # taken only words (not punctuation)
            token_words = [w for w in tokens if w.isalpha()]
            return token_words

        imdb['words'] = imdb.apply(identify_tokens, axis=1)
```

1.5 Stemming

Stemming reduces related words to a common stem. It is an optional process step, and it it is useful to test accuracy with and without stemming. Let's look at an example.

```
In [9]: from nltk.stem import PorterStemmer
        stemming = PorterStemmer()

        my_list = ['frightening', 'frightened', 'frightens']

        # Using a Python list comprehension method to apply to all words in my_list

        print ([stemming.stem(word) for word in my_list])
```

```
['frighten', 'frighten', 'frighten']
```

To apply this to all rows in our imdb DataFrame we will again define a function and apply it to our DataFrame.

```
In [10]: def stem_list(row):
             my_list = row['words']
             stemmed_list = [stemming.stem(word) for word in my_list]
             return (stemmed_list)

         imdb['stemmed_words'] = imdb.apply(stem_list, axis=1)
```

Lets check our stemmed words (using pandas DataFrame .iloc method to select the first row).

```
In [11]: print(imdb['stemmed_words'].iloc[0])
```

```
['i', 'have', 'no', 'read', 'the', 'novel', 'on', 'which', 'the', 'kite', 'runner', 'is', 'bas
```

1.6 Removing stop words

'Stop words' are commonly used words that are unlikely to have any benefit in natural language processing. These includes words such as 'a', 'the', 'is'.

As before we will define a function and apply it to our DataFrame.

We create a set of words that we will call 'stops' (using a set helps to speed up removing stop words).

```
In [12]: from nltk.corpus import stopwords
         stops = set(stopwords.words("english"))

         def remove_stops(row):
             my_list = row['stemmed_words']
             meaningful_words = [w for w in my_list if not w in stops]
             return (meaningful_words)

         imdb['stem_meaningful'] = imdb.apply(remove_stops, axis=1)
```

Show the stemmed words, without stop words, from the first record.

```
In [13]: print(imdb['stem_meaningful'][0])
```

```
['read', 'novel', 'kite', 'runner', 'base', 'wife', 'daughter', 'thought', 'movi', 'fell', 'lo
```

1.7 Rejoin meaningful stemmed words

Now we will rejoin our meaningful stemmed words into a single string.

```
In [14]: def rejoin_words(row):
             my_list = row['stem_meaningful']
             joined_words = ( " ".join(my_list))
             return joined_words

         imdb['processed'] = imdb.apply(rejoin_words, axis=1)
```

1.8 Save processed data

Now we'll save our processed data as a csv. We'll drop the intermediate columns in our Pandas DataFrame.

```
In [17]: print(list(imdb))
```

```
['review', 'sentiment', 'words', 'stemmed_words', 'stem_meaningful', 'processed']
```

```
In [18]: cols_to_drop = ['review', 'words', 'stemmed_words', 'stem_meaningful']
         imdb.drop(cols_to_drop, axis=1, inplace=True)

In [19]: imdb.to_csv('imdb_processed.csv', index=False)
```