03/01/2020

Feature Extraction with frequencies

→ Feature Extraction

$$X_m = [\,1,\ \sum_{w} freqs(w,1),\ \sum_{w} freqs(w,0)\,]$$

- X_m → features of tweet 'm'
- 1 → Bias
- $\sum_{w} freqs(w,1)$ → Sum of Pos. frequencies
- $\sum_{w} freqs(w,0)$ → Sum of Neg. frequencies

eg:-

→ I am sad, I am not learning NLP

$$X_m = [\,1,\ \sum_{w} freqs(w,1),\ \sum_{w} freqs(w,0)\,]$$

→ Now, if we look at the table - word frequencies in classes

then the data on which we have to look is

vocabulary	Pos. freq. (1)	Neg. freq.
I	3	3
am	3	3
happy	2	0
because	1	0
learning	1	1
NLP	1	1
sad	0	2
not	0	1

we don't have to look at these because in our example these words are not there

→ So,

$\sum_{w} freqs(w, 1) = 8$

$\sum_{w} freqs(w, 0) = 11$

→ $x_m = [1, 8, 11]$

Preprocessing: Stop words and punctuation

eg: → @YMourri and @AndrewYNG are tuning a GREAT AI model at https://deeplearning.ai!!!...

Stop words	Punctuation
and	,
is	.
are	:
at	;
has	!
for	"
a	'

→ Now, tweet will be

@YMourri and @AndrewYNG tuning GREAT AI model https://deeplearning.ai

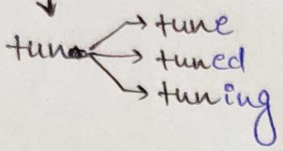
Preprocessing: Handles & URLs

→ ~~@YMourri and @AndrewYNG~~ tuning GREAT AI model ~~https://deeplearning.ai~~

⇒ tuning GREAT AI model

Preprocessing: stemming and lowercasing

tuning GREAT AI model



GREAT
Great
great
→ great

Preprocessed tweet:

[tun, great, ai, model]

EXERCISE

→ For, this exercise, we will use a twitter dataset that comes with NLTK.

→ About dataset

- The sample dataset from NLTK is separated into positive & negative tweets.
- It contains 5000 positive tweets and 5000 negative tweets exactly.

Import basic libraries

```
import nltk
from nltk.corpus import twitter_samples
import matplotlib.pyplot as plt
import random
```

→ Python library for NLP

→ sample twitter dataset from NLTK.

→ Library for visualization

→ pseudo-random number generators.

* To download twitter dataset at your local machine you can type:

→ nltk.download("twitter_samples")

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

→ all_positive_tweets = twitter_samples.strings('positive_tweets.json')

all_negative_tweets = twitter_samples.strings('negative_tweets.json')

→ print('Number of positive tweets:', len(all_positive_tweets))
↳ 5000

print('Number of negative tweets:', len(all_negative_tweets)) → 5000
↳ <class 'list'>

→ print('the type of all_positive tweets is:', type(all_negative_tweets[0]))
↳ <class 'str'>

→ # Declare a figure with a custom size

fig = plt.figure(figsize=(5,5))

Labels for the two classes

labels = 'Positive', '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()
```

→ # Print random positive & negative tweet in green & red.

```
print('\033[92m' + all_positive_tweets[random.randint(0,5000)])
print('\033[91m' + all_negative_tweets[random.randint(0,5000)])
```

↳ output:

i walked out of the fruit shop with LOTS of pumpkins & a guy smiled at me because it looked pretty funny, but doesn't matter, it worked :)

really didn't want to see that :-(

Preprocess raw text for sentiment analysis

→ The preprocessing steps are comprised of the following tasks:

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

→ # our selected sample. complex enough to exemplify each step.

```
tweet = all_positive_tweets[2277]
print(tweet)
```

→ # let's import few more library for this purpose.

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

→ import re → library for regular expression operations
import string → for string operations
from nltk.corpus import stopwords → module for stop words that comes with NLTK.
from nltk.stem import PorterStemmer → module for stemming
from nltk.tokenize import TweetTokenizer → module for tokenizing strings.

```
# 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 remove the hash # sign from the word)
tweet2 = re.sub(r'#', '', tweet2)
print(tweet2)
```

Tokenize the string

→
```
# Instantiate tokenizer class
tokenizer = TweetTokenizer(preserve_case=False,
                           strip_handles=True, reduce_len=True)
# Tokenize tweets
tweet_tokens = tokenizer.tokenize(tweets)
print('Tokenized string:\n' + tweet_tokens)
```

Remove stop words and punctuations

→
```
# 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)
```

→
```
tweets_clean = []
for word in tweet_tokens:  # Go through every word in your token 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 & punctuation:')
print(tweets_clean)
```

Stemming

→ # 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)
```

process_tweet()

→ As shown above, preprocessing consists of multiple steps before you arrive at the final list of words.

→ There is a function "process_tweet(tweet)" available in "utils.py" which perform all the same preprocessing steps which is discussed above.

→
```
from utils import process_tweet # Import process tweet function
# choose the same tweet
tweet = all_positive_tweets[2277]
print('\033[92m' + 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.
```