submitted (1)

January 27, 2022

1 Load train dataset

1. Using google files from colab, upload the training file, train.csv

[1]:
```
from google.colab import files
upload = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving train.csv to train.csv
```

2 Converting dataset from csv to pandas dataframe

1. Convert the uplaoded dataset into a pandas dataframe.

The variable df looks like

id	keyword	location	text	target
1	NaN	City A	That is a really bad cracker	0
4	Flood	City B	Floods after the rain RT #Flood @WaterBurst	1
5	Collapse	NaN	Big collapse in the half constructed building https://build	1
6	Fire	City C	Team is really fired up!	0
7	NaN	City D	Rocked	0

[2]:
```
import io
import pandas as pd
df = pd.read_csv(io.BytesIO(upload['train.csv']))
```

3 Clean Dataset

1. Replace NaN values to empty string, ' '.
2. Sort dataset according to target values

id	keyword	location	text	target
1		City A	That is a really bad cracker	0
6	Fire	City C	Team is really fired up!	0
7		City D	Rocked	0
4	Flood	City B	Floods after the rain RT #Flood @WaterBurst	1
5	Collapse		Big collapse in the half constructed building https://build	1

[3]:
```python
import numpy as np
df = df.sort_values('target')
df = df.fillna('')
```

4 Getting relevant columns (INPUT)

1. Get the keyword, text and location column

keyword	location	text
City A	That is a really bad cracker	
Fire	City C	Team is really fired up!
City D	Rocked	
Flood	B	Floods after the rain RT #Flood @WaterBurst
Collapse		Big collapse in the half constructed building https://build

2. Concate the strings element-wise

tweets
City A That is a really bad cracker
Fire City C Team is really fired up!
City D Rocked
Flood B Floods after the rain RT #Flood @WaterBurst
Collapse Big collapse in the half constructed building https://build

[4]:
```python
keyword = df['keyword']
location = df['location']
text = df['text']
```

[5]:
```python
tweets = np.add(keyword + ' ', np.add(text, ' ' + location))
```

5 Preprocess tweets function

1. Remove numbers

2. Remove special characters and any following that
3. Remove retweets
4. Remove links
5. Remove hashtags
6. Remove at the rates
7. Remove case to all small, strip handles, reduce the length using tokenizer
8. Remove stopwords
9. Stem the words
10. Remove punctuations
11. Store the clean data

tweets
['city', 'a', 'bad', 'cracker']
['fire', 'city', 'c', 'team', 'fire', 'up']
['city', 'd', 'rock']
['flood', 'city', 'b', 'flood', 'rain', 'flood', 'waterburst']
['collapse', 'big', 'collapse', 'half', 'construct', 'build']

[6]:
```
def process_tweet(tweet):
  tweet = re.sub(r"[0-9]", "", tweet)
  tweet = re.sub(r'\$\w*', '', tweet)
  tweet = re.sub(r'^RT[\s]+', '', tweet)
  tweet = re.sub(r'https?://[^\s\n\r]+', '', tweet)
  tweet = re.sub(r'http?://[^\s\n\r]+', '', tweet)
  tweet = re.sub(r'#', '', tweet)
  tweet = re.sub(r'@', '', tweet)
  tokenizer = TweetTokenizer(preserve_case=False,␣
↪strip_handles=True,reduce_len=True)
  tweet_tokens = tokenizer.tokenize(tweet)
  tweet_clean = []
  stopwords_english = stopwords.words('english')
  stemmer = PorterStemmer()
  for word in tweet_tokens:
    if word not in stopwords_english and word not in string.punctuation:
      stem_word = stemmer.stem(word)
      tweet_clean.append(stem_word)
  return tweet_clean
```

6 Get labels (ACTUAL OUTPUT)

1. Get the count of 0s and 1s usinf the Counter function.
2. Generate a numpy array of 0s and 1s.

labels
0
0
0
1
1

[7]:
```
from collections import Counter
target = df['target']
target_count = Counter(target)
labels = np.append(np.zeros((target_count[0], 1)), np.ones((target_count[1],␣
↪1)), axis = 0)
```

7 Getting frequency of each word function

1. Squeeze the labels to a list
2. Generate a dictionary with a tulple as key, (word, target_value) and increase its count

```
{
  ('city', 0): 4,
  ('a', 0): 1,
  ('bad', 0): 1,
  ('cracker', 0): 1,
  ('fire', 0): 2,
  ('c', 0): 1,
  ('team', 0): 1,
  ('up', 0): 1,
  ('d', 0): 1,
  ('rock', 0): 1,
  ('flood', 1): 3,
  ('b', 1): 1,
  ('rain', 1): 1,
  ('waterburst', 1): 1,
  ('collapse', 1): 2,
  ('big', 1): 1,
  ('half', 1): 1,
  ('construct', 1): 1,
  ('build', 1): 1
}
```

[8]:
```python
def build_freqs(tweets, ys):
    yslist = np.squeeze(ys).tolist()
    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
```

8 Using the above functions

[9]:
```python
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import TweetTokenizer
import string
import nltk
nltk.download('stopwords')
freqs = build_freqs(tweets, labels)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

9 Lookup function

Lookup function is used to find the frequency of a word given it is fake or real.

lookup(freqs, "city", 0) is 4

lookup(freqs, 'half', 1) is 1

[10]:
```python
def lookup(freqs, word, label):
    n = 0
    pair = (word, label)
    if (pair in freqs):
        n = freqs[pair]
    return n
```

10 Train Naive Bayes

vocab = [‘city’, ‘a’, ‘bad’, ‘cracker’, ‘fire’, ‘c’, ‘up’, ‘team’, ‘d’, ‘rock’, ‘flood’, ‘b’, ‘rain’, ‘waterburst’, ‘collapse’, ‘big’, ‘half’, ‘construct’, ‘build’]

V = 19

Number of positives = 12

Number of negatives = 14

D = 5

Number of positive labels = 2

Number of negative labels = 3

$$logprior = log\left(\frac{D_{pos}}{D_{neg}}\right)$$

The positive loglikelihood of a word, w is

$$\frac{freq_{pos,w_i} + 1}{N_{pos} + V}$$

The negative loglikelihood of a word, w is

$$\frac{freq_{neg,w_i} + 1}{N_{neg} + V}$$

The loglikelihood overall is the log of the ratio of the above

The loglikelihood of every word is given below

word	-ve likelihood	+ve likelihood	likelihood ratio	loglikelihood
‘city’	$\frac{4+1}{14+19}$	$\frac{0+1}{12+19}$	0.213	-1.540
‘a’	$\frac{1+1}{14+19}$	$\frac{0+1}{12+19}$	0.532	-0.631
‘bad’	$\frac{1+1}{14+19}$	$\frac{0+1}{12+19}$	0.532	-0.631
‘cracker’	$\frac{1+1}{14+19}$	$\frac{0+1}{12+19}$	0.532	-0.631
‘fire’	$\frac{2+1}{14+19}$	$\frac{0+1}{12+19}$	0.352	-1.044
‘c’	$\frac{1+1}{14+19}$	$\frac{0+1}{12+19}$	0.532	-0.631
‘team’	$\frac{1+1}{14+19}$	$\frac{0+1}{12+19}$	0.532	-0.631
‘up’	$\frac{1+1}{14+19}$	$\frac{0+1}{12+19}$	0.532	-0.631
‘d’	$\frac{1+1}{14+19}$	$\frac{0+1}{12+19}$	0.532	-0.631
‘rock’	$\frac{1+1}{14+19}$	$\frac{0+1}{12+19}$	0.532	-0.631
‘flood’	$\frac{0+1}{14+19}$	$\frac{3+1}{12+19}$	4.258	1.440
‘b’	$\frac{0+1}{14+19}$	$\frac{1+1}{12+19}$	2.129	0.755
‘rain’	$\frac{0+1}{14+19}$	$\frac{1+1}{12+19}$	2.129	0.755
‘waterburst’	$\frac{0+1}{14+19}$	$\frac{1+1}{12+19}$	2.129	0.755
‘collapse’	$\frac{0+1}{14+19}$	$\frac{2+1}{12+19}$	3.193	1.161

word	-ve likelihood	+ve likelihood	likelihood ratio	loglikelihood
'big'	$\frac{0+1}{14+19}$	$\frac{1+1}{12+19}$	2.129	0.755
'half'	$\frac{0+1}{14+19}$	$\frac{1+1}{12+19}$	2.129	0.755
'construct'	$\frac{0+1}{14+19}$	$\frac{1+1}{12+19}$	2.129	0.755
'build'	$\frac{0+1}{14+19}$	$\frac{1+1}{12+19}$	2.129	0.755

[11]:
```
def train_naive_bayes(freqs, train_x, train_y):
    loglikelihood = {}
    logprior = 0
    vocab = set(pair[0] for pair in freqs.keys())
    V = len(vocab)
    N_pos = N_neg = 0
    for pair in freqs.keys():
        if pair[1] > 0:
            N_pos += freqs[pair]
        else:
            N_neg += freqs[pair]
    D = len(train_y)
    D_pos = (len(list(filter(lambda x: x > 0, train_y))))
    D_neg = (len(list(filter(lambda x: x <= 0, train_y))))
    logprior = np.log(D_pos) - np.log(D_neg)
    for word in vocab:
        freq_pos = lookup(freqs,word,1)
        freq_neg = lookup(freqs,word,0)
        p_w_pos = (freq_pos + 1) / (N_pos + V)
        p_w_neg = (freq_neg + 1) / (N_neg + V)
        loglikelihood[word] = np.log(p_w_pos/p_w_neg)
    return logprior, loglikelihood
```

[12]:
```
logprior, loglikelihood = train_naive_bayes(freqs, tweets, labels)
```

11 Test data

Getting the weights and applying to the test data is to get the final value of the logistic regression.

$$\hat{y} = \begin{cases} 1 & y_{pred} \geq 0.0 \\ 0 & y_{pred} < 0.0 \end{cases} \tag{1}$$

Predicted value is after the weights value on the test data.

[13]:
```
from google.colab import files
upload = files.upload()
import io
import pandas as pd
```

```
df = pd.read_csv(io.BytesIO(upload['test.csv']))
df = df.fillna('')
keyword = df['keyword']
location = df['location']
text = df['text']
tweets = np.add(keyword + ' ', np.add(text + ' ', location))
```

<IPython.core.display.HTML object>

Saving test.csv to test.csv

[14]:
```
def naive_bayes_predict(tweet, logprior, loglikelihood):
    word_l = process_tweet(tweet)
    p = 0
    p += logprior
    for word in word_l:
        if word in loglikelihood:
            p += loglikelihood[word]
    return p
```

[15]:
```
def test_naive_bayes(test_x, test_y, logprior, loglikelihood, id,␣
 ↪naive_bayes_predict=naive_bayes_predict):
    accuracy = 0
    y_hats = []
    for tweet in test_x:
        if naive_bayes_predict(tweet, logprior, loglikelihood) > 0:
            y_hat_i = 1
        else:
            y_hat_i = 0
        y_hats.append(y_hat_i)
    id = list(id)
    import csv
    with open('submit.csv', 'w') as f:
      writer = csv.writer(f)
      writer.writerows(zip(['id'], ['target']))
      writer.writerows(zip(id, y_hats))
```

[16]:
```
id = df['id']
test_naive_bayes(tweets, labels, logprior, loglikelihood, id,␣
 ↪naive_bayes_predict=naive_bayes_predict)
```