FourthBrain

Hate Speech Detector

Today you are a machine learning engineer, a member of the Birdwatch at Twitter.

The objective of this task is to detect hate speech in tweets. For the sake of simplicity, a tweet contains hate speech if it has a racist or sexist sentiment associated with it. In other words, we need to classify racist or sexist tweets from other tweets.

A labelled dataset of 31,962 tweets (late 2017 to early 2018) is provided in the form of a compressed csv file with each line storing a tweet id, its label, and the tweet. Label '1' denotes the tweet is racist/sexist while label '0' denotes the tweet is not racist/sexist.

We will first approach the problem in a traditional approach: clean the raw text using simple regex (regular expression), extract features, build a naive Bayes models to classify tweets; then we build a deep learning model and explain our deep learning model with LIME.

📚 Learning Objectives

By the end of this lesson, you will be able to:

- Understand the basic concepts in natural language processing (NLP)
- Perform basic NLP tasks on text, e.g., tweets
- Build a naive Bayes classifier to detect hate speech
- Build a bidirectional long short-term memory (BiLSTM) to detect hate speech
- Visualize embeddings with Tensorboard embedding projector
- Explain models with LIME

Task I: Data Preprocessing

1. Install dependencies.

   Most modules are pre-installed in Colab, however, we need to update `gensim` to its recent version and install `lime`.

```
!pip install -U -q gensim==4.2.0 lime
```

```
     |████████████████████████████████| 24.1 MB 4.0 MB/s
     |████████████████████████████████| 275 kB 82.0 MB/s
  Building wheel for lime (setup.py) ... done
```

2. Connect Colab to your local Google Drive.

```
!nvidia-smi
```

```
Sat Oct 29 02:45:24 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  A100-SXM4-40GB      Off  | 00000000:00:04.0 Off |                    0 |
| N/A   32C    P0    41W / 400W |      0MiB / 40536MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

3. Use `pandas.read_csv` to load the tweets in `tweets.csv.gz` and save the `pd.DataFrame` into `raw`. Make sure the path points to where the data is located in your Google Drive.

```
# Avoids scrolling in Jupyter notebook cells
from IPython.display import Javascript
def resize_colab_cell():
  display(Javascript('google.colab.output.setIframeHeight(0, true, {maxHeight: 5000})'))
get_ipython().events.register('pre_run_cell', resize_colab_cell)
```

```
# YOUR CODE HERE
import pandas as pd

w10_path = 'drive/MyDrive/week-10-nlp/'
raw = pd.read_csv(w10_path + 'dat/tweets.csv.gz')
```

```
print(raw.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31962 entries, 0 to 31961
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      31962 non-null  int64 
 1   label   31962 non-null  int64 
 2   tweet   31962 non-null  object
dtypes: int64(2), object(1)
memory usage: 749.2+ KB
None
```

```
# let's look at the df:
max_tweet = 280    # max tweet length
pd.set_option('display.max_colwidth', 280)
raw
```

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run
1	2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in urð ±!!! ð ð ð ð ð ¦ð ¦ð ¦
4	5	0	factsguide: society now #motivation
...	...	...	...
31957	31958	0	ate @user isz that youuu?ð ð ð ð ð ð ð ð ð â ¤ï¸
31958	31959	0	to see nina turner on the airwaves trying to wrap herself in the mantle of a genuine hero like shirley chisolm. #shame #imwithher
31959	31960	0	listening to sad songs on a monday morning otw to work is sad
31960	31961	1	@user #sikh #temple vandalised in in #calgary, #wso condemns act
31961	31962	0	thank you @user for you follow

4. Sample 5 random tweets from the dataset for each label and display `label` and `tweet` columns. Hint: one option is to use `sample()` followed by `groupby`.

```
# let's take an overall look at the data:
raw.describe(include='all')
```

	id	label	tweet
count	31962.000000	31962.000000	31962
unique	NaN	NaN	29530
top	NaN	NaN	#model i love u take with u all the time in urð ±!!! ð ð ð ð ð ¦ð ¦ð ¦
freq	NaN	NaN	319
mean	15981.500000	0.070146	NaN
std	9226.778988	0.255397	NaN
min	1.000000	0.000000	NaN
25%	7991.250000	0.000000	NaN
50%	15981.500000	0.000000	NaN
75%	23971.750000	0.000000	NaN
max	31962.000000	1.000000	NaN

```
# sample 5 random tweets for ea label
cl0_cond = raw['label'] == 0.
cl1_cond = raw['label'] == 1.

display(raw[cl0_cond].sample(n=5))
display(raw[cl1_cond].sample(n=5))
```

	id	label	tweet
15097	15098	0	i get so excited when i see oppounities. i turn into a little kid that you just can't control myself. #joy #gotmelike #yippe
22088	22089	0	i live on the cape and haven't been to the beach once this summer :-) #work #please #save #me #idonthaveasunburn
19571	19572	0	details on website x #online #shopping #buy #gottohaveit #newâ ¦
23571	23572	0	that summer cut....#bnatural35 #shohairdontcare #tapered #naturalhair #natural #hair â ¦
14748	14749	0	the death of a dream is the day that you stop believing in the work it takes to get there. #dream #quoteoftheday

	id	label	tweet
23394	23395	1	#donaldtrump : i can't wait nothing good of a (racist) and #misogynist (misogynist) than see to the women like animals.
9416	9417	1	â ¡ï¸ â charles paladino's racist comments spark calls for resignationâ
18987	18988	1	@user you might be a libtard if... #libtard #sjw #liberal #politics
14270	14271	1	@user explain your actions. netanyahu you can not claim discrimination when u do it yourself to others.
10596	10597	1	#blacklivesmatter that's why bilal called the adhan on the kaaba always existed hamza yusuf that was not nice ð

```
# # Colab includes an extension that renders pandas dataframes into interactive displays that can be filtered, sorted, and explored dynamically.
# from google.colab import data_table
# data_table.enable_dataframe_formatter()
```

5. The tweets are in English and all words should be already in lowercase. Now calculate the number of characters in each tweet and assign the values to a new column `len_tweet` in `raw`.

```
# YOUR CODE HERE
raw['len_tweet'] = raw['tweet'].str.len()
display(raw)
```

	id	label	tweet	len_tweet
0	1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run	102
1	2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked	122
2	3	0	bihday your majesty	21
3	4	0	#model i love u take with u all the time in urð ±!!! ð ð ð ð ð ¦ð ¦ð ¦	86
4	5	0	factsguide: society now #motivation	39
...	...	...	...	...
31957	31958	0	ate @user isz that youuu?ð ð ð ð ð ð ð ð ð â ¤ï¸	68
31958	31959	0	to see nina turner on the airwaves trying to wrap herself in the mantle of a genuine hero like shirley chisolm. #shame #imwithher	131
31959	31960	0	listening to sad songs on a monday morning otw to work is sad	63
31960	31961	1	@user #sikh #temple vandalised in in #calgary, #wso condemns act	67
31961	31962	0	thank you @user for you follow	32

31962 rows × 4 columns

6. What are the summary statistics of `len_tweet` for each label? Hint: use `groupby` and `describe`.

```
pd.set_option("display.precision", 1)

# YOUR CODE HERE
raw.groupby('label').describe()
```

	id								**len_tweet**							
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
label																
0	29720.0	15974.5	9223.8	1.0	7981.8	15971.5	23965.2	31962.0	29720.0	84.3	29.6	11.0	62.0	88.0	107.0	274.0

Note we have an imbalanced dataset: the ratio of non-hate speech to hate speech is roughly 13:1.

7. Clean the tweets.

   We use `re` to perform basic text manipulations. Specically, remove anonymized user handle, remove numbers and special characters except hashtags.

```
import re
```

8. Remove user handles from the text in `tweet`, or anything directly following the symbols `@`, and save the resulting tweets to a new column `tidy_tweet` in `raw`.

   Hint: you can use `re.sub` on individual text and [`apply`]() a simple lambda function for the series `raw['tweet']`.

```
# Remove tweet user handles + anything following '@' and save cleaned v. to new col:
# set pattern to edit with re.sub:
pattern = '@[\w]*'

raw['tidy_tweet'] = raw['tweet'].apply(lambda x: re.sub(pattern, '', x))
raw.sample(5, random_state=203)
```

	id	**label**	**tweet**	**len_tweet**	**tidy_tweet**
790	791	1	@user and you keep telling that only aryans are allowed to rape women! you're just a troll! #eod @user @user	109	and you keep telling that only aryans are allowed to rape women! you're just a troll! #eod
21928	21929	0	@user what makes you ?	25	what makes you ?
25642	25643	0	â #nzd/usd extends rbnz-led rally, hits fresh 1-year high near 0.7150 #blog #silver #gold #forex	101	â #nzd/usd extends rbnz-led rally, hits fresh 1-year high near 0.7150 #blog #silver #gold #forex
20436	20437	0	i'm on a mission to ride all of the animals! #teamchanlv #vegas #lasvegas #funtimes â ¦	91	i'm on a mission to ride all of the animals! #teamchanlv #vegas #lasvegas #funtimes â ¦
22552	22553	0	the color of a human skin matters a lot to the system when it comes to judgement call.	88	the color of a human skin matters a lot to the system when it comes to judgement call.

9. Remove non-alphabetic characters yet keep symbols `#` from `tidy_tweet` and save the result in `tidy_tweet`. In other words, keep all 26 letters and `#`.

   Note: in some applications, punctuations, emojis, or whether the word is in all caps can be of use. You shall decide whether to extract such features for the application and perform error analysis to gain insight.

```
# Remove non-alphabetic characters but keep '#' in tidy_tweets and re-save:
# set pattern to edit with re.sub:
pattern = '[^a-zA-Z#]'

raw['tidy_tweet'] = raw['tidy_tweet'].apply(lambda x: re.sub(pattern, ' ', x))

# store a copy of this partially 'cleaned' tweet df for later use:
raw_semi_tidy = raw.copy(deep=True)

# take a look at some random samples at this stage of pre-processing:
raw.sample(5, random_state=203)
```

	id	**label**	**tweet**	**len_tweet**	**tidy_tweet**
790	791	1	@user and you keep telling that only aryans are allowed to rape women! you're just a troll! #eod @user @user	109	and you keep telling that only aryans are allowed to rape women you re just a troll #eod
21928	21929	0	@user what makes you ?	25	what makes you
25642	25643	0	â #nzd/usd extends rbnz-led rally, hits fresh 1-year high near 0.7150 #blog #silver #gold #forex	101	#nzd usd extends rbnz led rally hits fresh year high near #blog #silver #gold #forex
20436	20437	0	i'm on a mission to ride all of the animals! #teamchanlv #vegas #lasvegas #funtimes â ¦	91	i m on a mission to ride all of the animals #teamchanlv #vegas #lasvegas #funtimes
22552	22553	0	the color of a human skin matters a lot to the system when it comes to judgement call.	88	the color of a human skin matters a lot to the system when it comes to judgement call

10. Remove words that is shorter than 4 characters from the processed tweets.

    For example,

    `i m on a mission to ride all of the animals #teamchanlv #vegas #lasvegas #funtimes`

    will be reduced to

    `mission ride animals #teamchanlv #vegas #lasvegas #funtimes`

```
# Remove words < len=4 from tidy_tweets and re-save:

raw['tidy_tweet'] = raw['tidy_tweet'].apply(lambda x:
                        ' '.join([word for word in x.split() if len(word) > 3]))
raw.sample(5, random_state=203)
```

	id	**label**	**tweet**	**len_tweet**	**tidy_tweet**
790	791	1	@user and you keep telling that only aryans are allowed to rape women! you're just a troll! #eod @user @user	109	keep telling that only aryans allowed rape women just troll #eod
21928	21929	0	@user what makes you ?	25	what makes
25642	25643	0	â #nzd/usd extends rbnz-led rally, hits fresh 1-year high near 0.7150 #blog #silver #gold #forex	101	#nzd extends rbnz rally hits fresh year high near #blog #silver #gold #forex
20436	20437	0	i'm on a mission to ride all of the animals! #teamchanlv #vegas #lasvegas #funtimes â ¦	91	mission ride animals #teamchanlv #vegas #lasvegas #funtimes
22552	22553	0	the color of a human skin matters a lot to the system when it comes to judgement call.	88	color human skin matters system when comes judgement call

11. Remove stopwords and perform text normalization.

    We will use `stopwords` collection and `SnowballStemmer` in `nltk` for this task. Before doing so, we need to tokenize the tweets. Tokens are individual terms or words, and tokenization is simply to split a string of text into tokens. You can use [`str.split()`]() on individual text and [`apply`]() a simple lambda function for the series `raw['tidy_tweet']` and save the result into `tokenized_tweet`.

    Check out some methods for the built-in type `str` [here]().

```
tokenized_tweet = raw['tidy_tweet'].apply(lambda x: x.split())
tokenized_tweet.head()
```

```
0                [when, father, dysfunctional, selfish, drags, kids, into, dysfunction, #run]
1    [thanks, #lyft, credit, cause, they, offer, wheelchair, vans, #disapointed, #getthanked]
2                                                                     [bihday, your, majesty]
3                                                            [#model, love, take, with, time]
4                                                          [factsguide, society, #motivation]
Name: tidy_tweet, dtype: object
```

```
print(type(tokenized_tweet), len(tokenized_tweet), tokenized_tweet.shape)
```

```
<class 'pandas.core.series.Series'> 31962 (31962,)
```

12. Extract stop words and remove them from the tokens.

    Note: depending on the task / industry, it is highly recommended that one curate custom stop words.

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = stopwords.words('english')

print(type(stop_words), len(stop_words), '\n', *stop_words)
```

```
<class 'list'> 179
 i me my myself we our ours ourselves you you're you've you'll you'd your yours yourself yourselves he him his himself she she's her hers herself it it's its itself they them their theirs themsel
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
tokenized_tweet = tokenized_tweet.apply(lambda x: [word for word in x if
                                                   word not in stop_words])
```

```
assert any(word in tokenized_tweet for word in stop_words) == False
```

```
display(tokenized_tweet)
```

```
0                                              [father, dysfunctional, selfish, drags, kids, dysfunction, #run]
1                                 [thanks, #lyft, credit, cause, offer, wheelchair, vans, #disapointed, #getthanked]
2                                                                                                [bihday, majesty]
3                                                                                        [#model, love, take, time]
4                                                                                [factsguide, society, #motivation]
                                                      ...
31957                                                                                                     [youuu]
31958    [nina, turner, airwaves, trying, wrap, mantle, genuine, hero, like, shirley, chisolm, #shame, #imwithher]
31959                                                                        [listening, songs, monday, morning, work]
31960                                                            [#sikh, #temple, vandalised, #calgary, #wso, condemns]
31961                                                                                                [thank, follow]
Name: tidy_tweet, Length: 31962, dtype: object
```

13. Create a new instance of a language specific `SnowballStemmer`, set the `language` to be "english"; see how to.

```
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer('english')
```

```
tokenized_tweet = tokenized_tweet.apply(lambda x: [stemmer.stem(i) for i in x])
display(tokenized_tweet)
```

```
0                                          [father, dysfunct, selfish, drag, kid, dysfunct, #run]
1                              [thank, #lyft, credit, caus, offer, wheelchair, van, #disapoint, #getthank]
2                                                                                   [bihday, majesti]
3                                                                           [#model, love, take, time]
4                                                                       [factsguid, societi, #motiv]
                                                 ...
31957                                                                                         [youuu]
31958    [nina, turner, airwav, tri, wrap, mantl, genuin, hero, like, shirley, chisolm, #shame, #imwithh]
31959                                                               [listen, song, monday, morn, work]
31960                                                    [#sikh, #templ, vandalis, #calgari, #wso, condemn]
31961                                                                                  [thank, follow]
Name: tidy_tweet, Length: 31962, dtype: object
```

14. Lastly, let's stitch these tokens in `tokenized_tweet` back together and save them in `raw['tidy_tweet']`. Use `str.join()` and `apply`.

```
raw['tidy_tweet'] = tokenized_tweet.apply(lambda x: ' '.join(x))
display(raw)
```

	id	label	tweet	len_tweet	tidy_tweet
0	1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run	102	father dysfunct selfish drag kid dysfunct #run
1	2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked	122	thank #lyft credit caus offer wheelchair van #disapoint #getthank
2	3	0	bihday your majesty	21	bihday majesti
3	4	0	#model i love u take with u all the time in urð ±!!! ð ð ð ð ð ¦ð ¦ð ¦	86	#model love take time
4	5	0	factsguide: society now #motivation	39	factsguid societi #motiv
...	...	...	...	...	...
31957	31958	0	ate @user isz that youuu?ð ð ð ð ð ð ð ð ð â ¤ï¸	68	youuu
31958	31959	0	to see nina turner on the airwaves trying to wrap herself in the mantle of a genuine hero like shirley chisolm. #shame #imwithher	131	nina turner airwav tri wrap mantl genuin hero like shirley chisolm #shame #imwithh
31959	31960	0	listening to sad songs on a monday morning otw to work is sad	63	listen song monday morn work
31960	31961	1	@user #sikh #temple vandalised in in #calgary, #wso condemns act	67	#sikh #templ vandalis #calgari #wso condemn
31961	31962	0	thank you @user for you follow	32	thank follow

31962 rows × 5 columns

Task 2. Wordcloud and Hashtag

In this task, we want to gain a general idea of what the common words were and how hashtags were used in tweets. We will create wordclouds and extract the top hashtags used in each label.

1. Before doing so, out of caution of possible data leakage, split the `raw['tidy_tweet']` into training and test datasets in a stratified fashion, set the test size at .25 and random state as 42.

   Save the results into `X_train`, `X_test`, `y_train`, and `y_test`.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    raw['tidy_tweet'], raw.label,
    test_size=0.25, random_state=42, stratify=raw.label)
```

```
assert X_train.shape == y_train.shape == (23971, )
assert X_test.shape == y_test.shape == (7991,)
```

```
# verify label split in train & test comparable:
print(f'In y_train, ratio of label 0 to 1 is: '
      f'{y_train.value_counts()[0]/y_train.value_counts()[1]:.4f} : 1, while\n'
      f'In y_test , ratio of label 0 to 1 is: '
      f'{y_test.value_counts()[0]/y_test.value_counts()[1]:.4f} : 1')
```

```
In y_train, ratio of label 0 to 1 is: 13.2600 : 1, while
In y_test , ratio of label 0 to 1 is: 13.2442 : 1
```

2. A word cloud is a cluster of words depicted in different sizes. The bigger the word appears, the more often it appears in the given text. It can offer an easy visual presentation to reveal the theme of a topic.

   Function `plot_wordcloud` is provided to plot 50 most frequent words from the given text in the shape of twitter's logo. You may need to replace the image path accordingly.

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image

def plot_wordcloud(text:str) -> None:
    '''
    Plot a wordcloud of top 50 words from the input text
    masked by twitter logo
    '''
    mask = np.array(Image.open(w10_path + 'img/twitter-mask.png')) # REPLACE w/ YOUR FILE PATH
    wordcloud = WordCloud(
        background_color='white',
        random_state=42,
        max_words=50,
        max_font_size=80,
        mask = mask).generate(text)
    plt.figure(figsize=(10,10))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.axis("off")
    plt.show()
```

3. Visualize the wordcloud.

   The function expects one long string. Stitch all tidy tweets from training set and save the single string to `all_words`, then visualize the wordcloud for all the words.

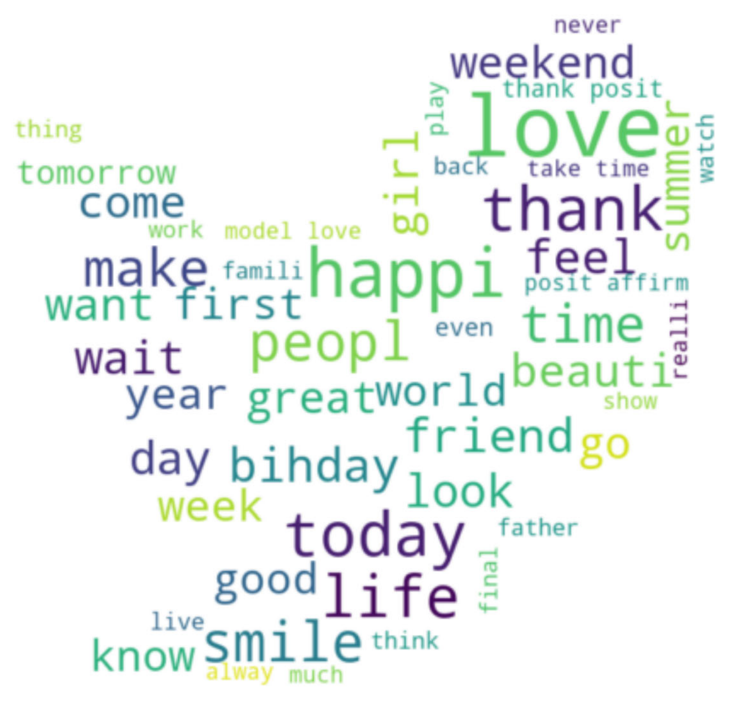
```
print(type(X_train), type(X_train[0]), '\n', X_train[0], '\n', X_train[2400])
```

```
<class 'pandas.core.series.Series'> <class 'str'>
 father dysfunct selfish drag kid dysfunct #run
 hour uncl sayeth disturb
```

```
# stitch all tidy_tweet words from X_train:
all_words = ' '.join(X_train)
print(type(all_words), len(all_words), '\n', all_words[:5000])

# confirm above:
all_words_l = ''

for _, values in X_train.items():
    all_words_l += ' ' + values

# (remove space in front of the very first word)
all_words_l = all_words_l[1:]
print(type(all_words_l), len(all_words_l), '\n', all_words_l[:5000])
```

```
<class 'str'> 1138271
 like spread peanut butter white bread #littlewond watch made america simpson interest went still go franci underwood seen leav marseill #nojok #enjoy #music #today #free #app #free #music #juic
<class 'str'> 1138271
 like spread peanut butter white bread #littlewond watch made america simpson interest went still go franci underwood seen leav marseill #nojok #enjoy #music #today #free #app #free #music #juic
```

```
plot_wordcloud(all_words)
```



4. Visualize the wordcloud just for the text from the tweets identified as hate speech.

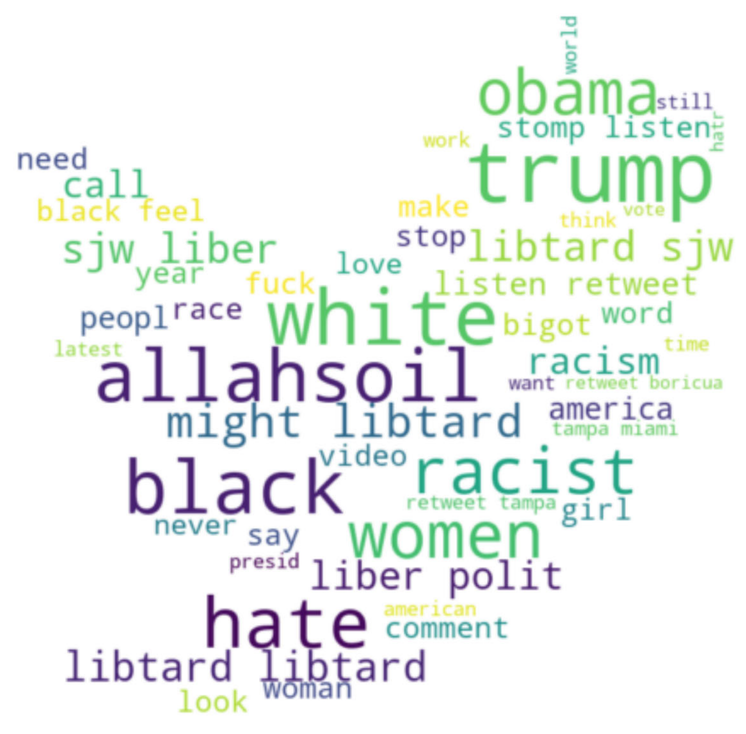
   Similarly, you need to stitch all the tidy tweets in training set that were identified as hate speech. Save the long string to `negative_words`.

```
# from the TRAINING SET X_train:
# stitch all tidy tweets identified as hate speech, and at the same time,
# stitch all tidy tweets NOT identified as hate speech (i.e., positive_words =
# all_words - negative_words)
negative_words = ''
positive_words = ''
non_labeled_words = ''

```

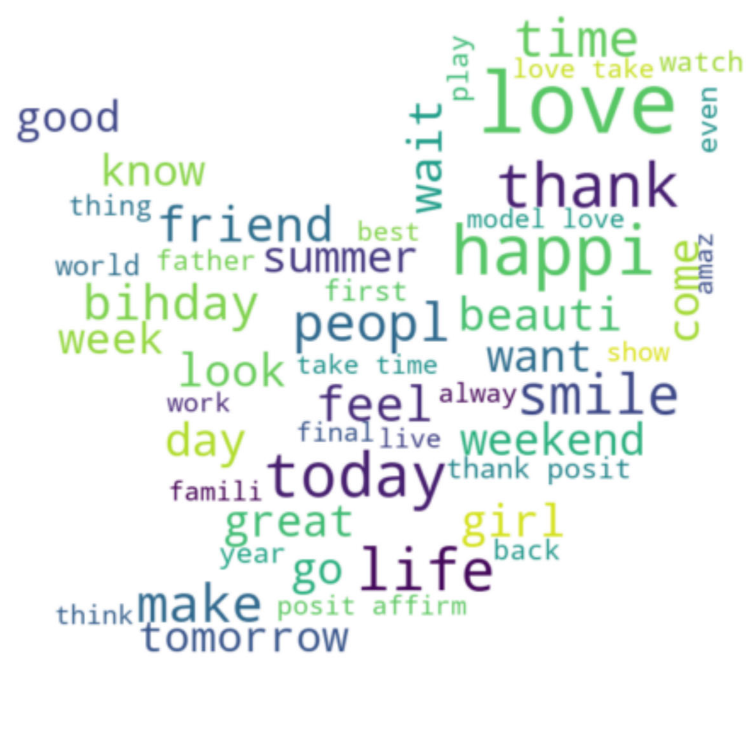
```
for index, value in y_train.items():
    if value == 1:
        negative_words += ' ' + X_train[index]
    elif value == 0:
        positive_words += ' ' + X_train[index]
    else:
        non_labeled_words += ' ' + X_train[index]

# remove space in front of very first word of above assembled strings (= an
# artefact of the above assembly method):
negative_words = negative_words[1:]
positive_words = positive_words[1:]
if len(non_labeled_words) > 0:
    non_labeled_words = non_labeled_words[1:]

# split assembled strings into lists of words (for counting whole words & also later):
neg_words_list = negative_words.split()
pos_words_list = positive_words.split()
all_words_list = all_words.split()

# show info for assembled strings at CHAR level:
print('',type(negative_words),len(negative_words),'\n',negative_words[:500],'\n\n',
      type(positive_words),len(positive_words),'\n',positive_words[:500], '\n\n',
      type(non_labeled_words),len(non_labeled_words),'\n',non_labeled_words[:])

print(f'The percentage of characters & spaces from \n\t'
      f'hate tweets is: \t'
      f'{len(negative_words)/len(all_words):.4%}\n')

# show info for assembled strings at WORD level:
print(f'The no. of negative words is: \t{len(neg_words_list):7d}\n'
      f'The no. of positive words is: \t{len(pos_words_list):7d}\n'
      f'The no. of total words is:    \t{len(all_words_list):7d}\n\n'
      f'The percentage of words that are \n\t'
      f'from neg. tweets is: \t'
      f'{(len(neg_words_list)/len(all_words_list)):.4%}')
```

```
 <class 'str'> 86223
 fuck fuck bitch real simpleton figur pmet youth parliament #democraci listen free prop #jesus #black white promot #hate #biggotri need call vile scum fed provid inadequ #mentalhealth care #first

 <class 'str'> 1052047
 like spread peanut butter white bread #littlewond watch made america simpson interest went still go franci underwood seen leav marseill #nojok #enjoy #music #today #free #app #free #music #juic

 <class 'str'> 0

The percentage of characters & spaces from
        hate tweets is:         7.5749%

The no. of negative words is:     12001
The no. of positive words is:    149100
The no. of total words is:       161101

The percentage of words that are
        from neg. tweets is:    7.4494%
```

```
plot_wordcloud(negative_words)    # (from hate tweets in training set)
```



```
# let's also look at the word cloud for only positive tweets:
plot_wordcloud(positive_words)    # (from non-hate tweets in training set)
```



5. Hashtag is a feature for tweets and we would like to inspect if hashtags provide information for our classification task.

   Function `hashtag_extract` is provided to extract hastags from an iterable (list or series) and return the hashtags in a list.

```
def hashtag_extract(x) -> list:
    """
    extract hastags from an iterable (list or series) and
    return the hashtags in a list.
    """
    hashtags = []
    # Loop over the words in the tweet
    for i in x:
        ht = re.findall(r"#(\w+)", i)
        hashtags.append(ht)
    return hashtags
```

6. Extract hashtags from ***non-hate*** speech tweets.

```
# from TRAINING SET X_train:
# NB: since we're extracting hashtags from non-hate speech tweets, we need to
# look at pos_words_list and NOT all_words_list!

HT_regular = hashtag_extract(pos_words_list)

# take a look at HT_regular:
print(len(HT_regular))
print(HT_regular[:24])
print(HT_regular[6], HT_regular[20])
```

```
149100
[[], [], [], [], [], [], ['littlewond'], [], [], [], [], [], [], [], [], [], [], [], [], [], ['nojok'], ['enjoy'], ['music'], ['today']]
['littlewond'] ['nojok']
```

```
assert type(HT_regular) == list
assert type(HT_regular[0]) == list # nested list
```

7. Now extract hashtags from hate speech tweets.

```
# from TRAINING SET X_train:
HT_negative = hashtag_extract(neg_words_list)
print(len(HT_negative))
```

```
12001
```

8. Both `HT_regular` and `HT_negative` are nested lists, so use the following trick to unnest both lists.

```
HT_regular = sum(HT_regular,[])
HT_negative = sum(HT_negative,[])

# take a look:
print(len(HT_regular), len(HT_negative))
print(HT_regular[:16], '\n', HT_negative[:16])
```

```
51324 3466
['littlewond', 'nojok', 'enjoy', 'music', 'today', 'free', 'app', 'free', 'music', 'juic', 'notsobad', 'healthyliv', 'eatclean', 'fit', 'cook', 'yummi']
 ['democraci', 'jesus', 'black', 'hate', 'biggotri', 'mentalhealth', 'firstnat', 'thirdworldcanada', 'mmusimaiman', 'zuma', 'malema', 'southafrica', 'hate', 'gop', 'con', 'nazi']
```

```
assert type(HT_regular) == type(HT_negative) == list
assert type(HT_regular[0]) == type(HT_negative[0]) == str
```

9. Complete the function `top_hashtags` below to take a list of hashtags and return the top `n` hashtag keyword and its frequncy.

```
from typing import List, Tuple
from collections import Counter
def top_hashtags(hashtags:List[str], n=10) -> List[Tuple[str, int]]:
    ''' Function to return the top n hashtags '''
    # YOUR CODE HERE
    # pass hashtag list to an instance of Counter
    my_Counter = Counter(hashtags)

    # return n most common hashtags
    return my_Counter.most_common(n)
```

1. Apply the function to the hashtag lists from the non-hate speech tweets and the hate speech tweets.

```
# YOUR CODE HERE
# 12 most common hashtages from non-hate speech and hate-speech
# tweets in TRAINING SET, X_train:
top_reg_hashtags = top_hashtags(HT_regular, 12)
top_negative_hashtags = top_hashtags(HT_negative, 12)
```

```
# YOUR CODE HERE
# let's take a look:
print(top_reg_hashtags)
print(top_negative_hashtags)
```

```
[('love', 1245), ('posit', 693), ('smile', 514), ('healthi', 423), ('thank', 398), ('fun', 345), ('life', 329), ('affirm', 324), ('summer', 291), ('model', 284), ('beauti', 273), ('friend', 269)]
[('trump', 97), ('allahsoil', 77), ('polit', 73), ('liber', 62), ('libtard', 57), ('sjw', 56), ('retweet', 50), ('miami', 37), ('black', 36), ('hate', 28), ('bigot', 27), ('tampa', 26)]
```

10. DISCUSS: are these hashtags making sense? should we include them as features or should we strip the # before tokenizing (that is, treat "#love" the same as "love")? why and why not?

Answer:

Hashtags are chosen *by the tweeter* presumably as *topics/concepts* they regard as ***especially relevant*** to their tweet:

- As such, they are likely of ***greater importance*** than any individual, non-hashtagged words in a tweet.
- Looking at the top 12 most frequently used hashtags among non-hate and hate tweets, we see that the tagged words *make sense, viz.* they represent topics/concepts that *we might expect to most commonly find* in tweets of their respective categories.
- From the above data, we can see that hashtags constitute a ***significant portion*** of the tweets in our training set `X_train`:
    - In non-hate tweets, there are 51,324 hashtags among a total of 149,100 words or **34%!**
    - In hate tweets, there are 3,466 hashtags among 12,001 words or **29%!**

In short, the above observations suggest that:

- We should ***certainly include*** hashtags as features; and,
- We should ***strongly consider*** retaining the ***# symbol*** since the hashtags likely represent ***ideas/concepts of particular importance*** to the tweeter, and should thus be weighed ***more heavily*** in our sentiment analysis model.

Task 3. Features

Note that almost all the machine learning related Python modules expect numerical presentation of data; thus we need to transform our text first. We will experiment with bag of words, tf-idf, and word2vec.

1. Convert the collection of text documents to a matrix of token counts.

Check the official documentation.

Create an instance of `CountVectorizer` named `bow_vectorizer`, set `max_features` to be `MAX_FEATURES`. Learn the vocabulary dictionary and return document-term matrix and save it to `bow_train`. Use `.fit_transform`.

```
from sklearn.feature_extraction.text import CountVectorizer

MAX_FEATURES = 1000

# stop_words='english' parameter not incl. here, since earlier,
# we used nltk to remove these!
bow_vectorizer = CountVectorizer(max_features=MAX_FEATURES)
bow_train = bow_vectorizer.fit_transform(X_train)
```

```
assert bow_train.shape == (X_train.shape[0], MAX_FEATURES)
```

2. Print the first three rows from `bow_train`. Hint: `.toarray()`.

```
print(type(bow_train))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
```

```
# YOUR CODE HERE
# Before looking at bow_train, get a dense ndarray of this sparse.csr_matrix:
dense_bow_train = bow_train.toarray()
print(type(dense_bow_train), dense_bow_train.shape)

# look at first 3 rows:
print(dense_bow_train[:3,:])
```

```
<class 'numpy.ndarray'> (23971, 1000)
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

```
from scipy.sparse.csr import csr_matrix
assert type(bow_train) == csr_matrix
```

3. Similarly, convert the collection of text documents to a matrix of TF-IDF features.

Create an instance of `TfidfVectorizer` named `tfidf_vectorizer`, set `max_features` to be `MAX_FEATURES`.

Learn the vocabulary and idf, return document-term matrix and save it to `tfidf_train`.

```
from sklearn.feature_extraction.text import TfidfVectorizer
# stop_words='english' parameter removed, since earlier, we used
# nltk to remove these!

tfidf_vectorizer = TfidfVectorizer(max_features=MAX_FEATURES)
tfidf_train = tfidf_vectorizer.fit_transform(X_train)
```

```
assert type(tfidf_train) == csr_matrix
assert tfidf_train.shape == bow_train.shape == (X_train.shape[0], MAX_FEATURES)
```

4. Extract word embeddings using Word2Vec. We will use `gensim` for this task.

The Word2Vec model takes either a list of lists of tokens or an iterable that streams the sentences directly from disk/network. Here, we tokenize the tidy tweets in `X_train` and save the list (`pd.series`) of lists of tokens to `tokenized_tweet`.

```
# Tokenize tweets in tidy_tweet in TRAINING X_train set:
tokenized_tweet = X_train.apply(lambda x: x.split())
print(type(tokenized_tweet), tokenized_tweet.shape)
```

```
<class 'pandas.core.series.Series'> (23971,)
```

```
assert tokenized_tweet.shape == X_train.shape
```

```
tokenized_tweet.head()
```

```
1036                                    [like, spread, peanut, butter, white, bread, #littlewond]
2380                                   [watch, made, america, simpson, interest, went, still, go]
31605                                          [franci, underwood, seen, leav, marseill, #nojok]
23437                                        [#enjoy, #music, #today, #free, #app, #free, #music]
2669     [#juic, experi, #notsobad, #healthyliv, #eatclean, #fit, #cook, #yummi, #mamasgotanewtoy]
Name: tidy_tweet, dtype: object
```

5. Import `Word2Vec` from `gensim.models`; see doc.

Create a skip-gram `Word2Vec` instance named `w2v` that learns on the `tokenized_tweet`, with `vector_size` set at `MAX_FEATURES`, and other parameters are provided.

```
from gensim.models import Word2Vec
```

```
# create skip-gram Word2Vec instance of above tokenized_tweet derived from
# TRAINING X_train set:
w2v = Word2Vec(
        sentences=tokenized_tweet,
        vector_size=MAX_FEATURES,
        window=5, min_count=2, sg = 1,
        hs = 0, negative = 10,  workers= 2,
        seed = 34)
```

6. Train the skip-gram model, set the epochs at 20.

```
print(type(w2v))
```

```
<class 'gensim.models.word2vec.Word2Vec'>
```

```
# To print ongoing w2v training log-loss after each epoch:
# Implements 'get_latest_training_loss()' method of Word2Vec model.train,
# which is enabled by setting 'compute_loss' parameter of model.train=True;
# (modified from ref[1] below):
from gensim.test.utils import common_texts, get_tmpfile
from gensim.models.callbacks import CallbackAny2Vec

class callback(CallbackAny2Vec):
    '''Callback to print loss after each epoch.'''

    def __init__(self):
        self.epoch = 0
        self.loss_to_sub = 0
        self.loss_list = []

    def on_epoch_end(self, model):
        loss = model.get_latest_training_loss()
        loss_now = loss - self.loss_to_sub
        self.loss_list.append(loss_now)
        print(f'Log-loss cum.: {loss:8.0f};   @ epoch {self.epoch:2d}:   {loss_now:.0f};')

        if self.epoch >= 1:
            print(f'\t\t\t\tΔ loss: '
                f'{(self.loss_list[self.epoch] - self.loss_list[self.epoch-1]):6.0f} '
                f'= {((self.loss_list[self.epoch]-self.loss_list[self.epoch-1]) / self.loss_list[self.epoch-1]):.2%}')
        self.loss_to_sub = loss
        self.epoch += 1
```

```
%%time
# YOUR CODE HERE
# TRAIN Word2Vec model on tokenized_tweet from X_train for 20 epochs,
# printing cumulative log-loss after each epoch:
w2v.train(
    tokenized_tweet,
    total_examples=tokenized_tweet.shape[0],
    epochs=20,
    word_count=0,
    compute_loss=True,
    callbacks=[callback()]  # an iterable of 'CallbackAny2Vec'; see above cell
)
```

```
WARNING:gensim.models.word2vec:Effective 'alpha' higher than previous training cycles
Log-loss cum.:   740234;   @ epoch  0:   740234;
Log-loss cum.:  1445523;   @ epoch  1:   705290;
                                Δ loss: -34944 = -4.72%
Log-loss cum.:  2127048;   @ epoch  2:   681524;
                                Δ loss: -23765 = -3.37%
Log-loss cum.:  2726260;   @ epoch  3:   599213;
                                Δ loss: -82311 = -12.08%
Log-loss cum.:  3307420;   @ epoch  4:   581159;
                                Δ loss: -18054 = -3.01%
Log-loss cum.:  3873151;   @ epoch  5:   565732;
                                Δ loss: -15428 = -2.65%
Log-loss cum.:  4402416;   @ epoch  6:   529265;
                                Δ loss: -36466 = -6.45%
Log-loss cum.:  4883612;   @ epoch  7:   481195;
                                Δ loss: -48070 = -9.08%
Log-loss cum.:  5353965;   @ epoch  8:   470354;
                                Δ loss: -10842 = -2.25%
Log-loss cum.:  5814919;   @ epoch  9:   460954;
                                Δ loss:  -9400 = -2.00%
Log-loss cum.:  6268018;   @ epoch 10:   453099;
                                Δ loss:  -7855 = -1.70%
Log-loss cum.:  6712322;   @ epoch 11:   444304;
                                Δ loss:  -8796 = -1.94%
Log-loss cum.:  7149620;   @ epoch 12:   437298;
                                Δ loss:  -7006 = -1.58%
Log-loss cum.:  7576848;   @ epoch 13:   427229;
                                Δ loss: -10069 = -2.30%
Log-loss cum.:  7999669;   @ epoch 14:   422820;
                                Δ loss:  -4408 = -1.03%
Log-loss cum.:  8413630;   @ epoch 15:   413961;
                                Δ loss:  -8860 = -2.10%
Log-loss cum.:  8787180;   @ epoch 16:   373550;
                                Δ loss: -40411 = -9.76%
Log-loss cum.:  9154208;   @ epoch 17:   367028;
                                Δ loss:  -6522 = -1.75%
Log-loss cum.:  9519927;   @ epoch 18:   365719;
                                Δ loss:  -1309 = -0.36%
Log-loss cum.:  9881581;   @ epoch 19:   361654;
                                Δ loss:  -4065 = -1.11%
CPU times: user 1min 8s, sys: 80.8 ms, total: 1min 8s
Wall time: 34.3 s
(2736628, 3222020)
```

7. Let's see how the model performs. Specify a word and print out the 10 most similar words from the our tweets in the training set. Use `most_similar`. Hint: print the type of `w2v` and `w2v.wv`.

```
# YOUR CODE HERE
print(type(w2v), type(w2v.wv))

# after training skip-gram w2v model on X_train, get keyed vectors:
word_vectors=w2v.wv

# get top 10 most similar words in X_train to specified query words:
query_words = ['woman', 'man', 'kid', 'gentleman', 'asian', '#love', 'love',    \
               '#posit', 'posit', '#smile', 'smile', '#healthi', 'healthi',     \
               '#thank', 'thank', '#trump', 'trump', '#allahsoil', \
               '#polit', 'polit', '#liber', 'liber', '#sjw', '#retweet', \
               'retweet', '#miami', 'miami', '#black', 'black']
top_n = 10

for query_word in query_words:
    query_sim = word_vectors.most_similar(positive=[query_word], topn=top_n)
    print(f'\nQuery word: {query_word}\n'
          f'Top {top_n} sim words in X_train tweets by cos sim (word, sim score):')
    print(*[top_words for top_words in query_sim], sep='\n')
```

```
<class 'gensim.models.word2vec.Word2Vec'> <class 'gensim.models.keyedvectors.KeyedVectors'>

Query word: woman
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('condescens', 0.5987967848777771)
('cautious', 0.5725727677345276)
('shiless', 0.5351705551147461)
('shag', 0.5348032116889954)
('slur', 0.5326204299926758)
('korean', 0.5316252112388611)
('malign', 0.5273106694221497)
('#rondarousey', 0.526350200176239)
('#awkward', 0.5216630101203918)
('prospect', 0.5163248777389526)

Query word: man
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('elsewher', 0.763234555721283)
('ceifi', 0.7214909791946411)
('merri', 0.7100830078125)
('hop', 0.7024528980255127)
('yeahhh', 0.7001987099647522)
('abroad', 0.6985730528831482)
('tribal', 0.6980041265487671)
('blackmail', 0.6973356604576111)
('austin', 0.6940761208534241)
('#arsen', 0.6938982605934143)

Query word: kid
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('dentist', 0.6189342141151428)
('irrespons', 0.6188347339630127)
('dysfunct', 0.591941237449646)
('#ece', 0.5767194628715515)
('cope', 0.5671834945678711)
('swap', 0.5511450171470642)
('wohless', 0.548711895942688)
('#poopin', 0.5480124950408936)
('#psych', 0.5459977984428406)
('teacher', 0.5431080460548401)

Query word: gentleman
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('filthi', 0.8453999757766724)
('bham', 0.8398491144180298)
('#notbetterweath', 0.8372629284858704)
('#ist', 0.8240733742713928)
('#biggestfan', 0.8234021663665771)
('faustus', 0.8231794238090515)
('#summershow', 0.8221796154975891)
('dipshit', 0.8214355707168579)
('yesssss', 0.8193938136100769)
('#beyourownboss', 0.8175444006919861)

Query word: asian
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('malign', 0.7050723433494568)
('bewar', 0.682373046875)
('edinburgh', 0.6779972314834595)
('avers', 0.6723843216896057)
('token', 0.6578561663627625)
('overdos', 0.6570450067520142)
('fetish', 0.6554417610168457)
('leicest', 0.6534872055053711)
('#pjk', 0.652367115020752)
('histor', 0.6492588520050049)

Query word: #love
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#bffs', 0.5270839333534241)
('#cruis', 0.5145883560180664)
('#soulmat', 0.5008472800254822)
('#forev', 0.4990968704223633)
('#favor', 0.4960387945175171)
('#moment', 0.49489539861679077)
('#canin', 0.49320167303085327)
('#lift', 0.48953986167907715)
('#hotti', 0.48926576972007751)
('#mama', 0.4877878427505493)

Query word: love
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('grant', 0.4785802662372589)
('#alreadi', 0.4752189517021179)
('rejoic', 0.4602426290512085)
('joshua', 0.4587012529373169)
('maddi', 0.4584828019142151)
('teamwork', 0.45838913321495056)
('iloveyou', 0.45798054337501526)
('beaut', 0.4550124704837799)
('#what', 0.4514105021953583)
('thanku', 0.4493773281574249)

Query word: #posit
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('flop', 0.6897341012954712)
('#thank', 0.6671457886695862)
('flip', 0.6480416059494019)
('vivaci', 0.6465914845466614)
('#affirm', 0.6457162499427795)
('wholesom', 0.6445557475090027)
('enchant', 0.6327435374259949)
('farmer', 0.6324706673622131)
('reassur', 0.6311824321746826)
('glamor', 0.6263628005981445)

Query word: posit
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('reap', 0.6429829597473145)
('attitud', 0.6372398734092712)
('#bentley', 0.6267654299736023)
('queenaneliss', 0.6232290267944336)
('miriam', 0.5994590520858765)
('energet', 0.5921592116355896)
('negat', 0.5748235583305359)
('#listento', 0.5741627812385559)
('#areacodemixshow', 0.5733129382133484)
('blown', 0.5731226801872253)

Query word: #smile
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#smiley', 0.5894156098365784)
('#teeth', 0.561453640460968)
('#beautifulsmil', 0.5592922568321228)
('#mileycyrus', 0.5533548593521118)
('#greeney', 0.5457953214645386)
('#selca', 0.5453092455863953)
('#dimpl', 0.5450261235237122)
```

```
('#keepsmil', 0.5442640781402588)
('#beaut', 0.540033757686615)
('#instahappi', 0.5399436950683594)

Query word: smile
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('giggl', 0.5559498071670532)
('#dogslif', 0.5410833954811096)
('#teeth', 0.5385761260986328)
('#dogfriend', 0.5297390222549438)
('#companymusicvideo', 0.529674232006073)
('mindsconsol', 0.5282706618309021)
('smiley', 0.5271536111831665)
('contagi', 0.5243247151374817)
('#beaut', 0.5242890119552612)
('reciproc', 0.5239123106002808)

Query word: #healthi
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#weightloss', 0.6443475484848022)
('#altwaystoh', 0.6191030740737915)
('homeopath', 0.6112611293792725)
('#herbalremedi', 0.6001065373420715)
('#empower', 0.5964471697807312)
('#nutrit', 0.5962268114089966)
('#lawofattract', 0.593503475189209)
('herbal', 0.592623233795166)
('#iamchoosinglov', 0.5905634164810181)
('#joytrain', 0.5885463953018188)

Query word: healthi
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('immun', 0.7022883296012878)
('pursu', 0.692726731300354)
('#heatherfitzpatrick', 0.6902657747268677)
('lean', 0.6857295036315918)
('accomplish', 0.6856513023376465)
('#halthi', 0.6811622977256775)
('#doplant', 0.6588199138641357)
('#askmehow', 0.6546969413757324)
('#interest', 0.6504548788070679)
('shake', 0.6468455195426941)

Query word: #thank
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('flop', 0.6853679418563843)
('#posit', 0.6671457886695862)
('flip', 0.6454150676727295)
('thank', 0.625449001789093)
('farmer', 0.6234410405158997)
('freezer', 0.6013306379318237)
('grandpar', 0.5872949361801147)
('youuu', 0.5846593976020813)
('#igersbatanga', 0.584333598613739)
('pajama', 0.5814643502235413)

Query word: thank
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#thank', 0.6254488825798035)
('flop', 0.5573205947875977)
('glte', 0.5276926159858704)
('positivementalattitud', 0.5261961221694946)
('flip', 0.5087863206863403)
('tast', 0.5046872496604919)
('farmer', 0.49673691391944885)
('freezer', 0.49429890513420105)
('#motoz', 0.4942523241043091)
('blicqer', 0.49194177985191345)

Query word: #trump
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('lifter', 0.6741067171096802)
('welfar', 0.6671666502952576)
('melania', 0.6624009609222412)
('#biher', 0.6512031555175781)
('mustfal', 0.640261173248291)
('#swastika', 0.6384928226470947)
('#republican', 0.6366431713104248)
('#uselect', 0.6188860535621643)
('inaugur', 0.6184219121932983)
('#christmasev', 0.6172634363174438)

Query word: trump
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('donald', 0.6415885090827942)
('croni', 0.5825759768486023)
('#makeamericagreatagain', 0.5785855650901794)
('unfit', 0.576225221157074)
('commi', 0.5759557485580444)
('unstabl', 0.5746222734451294)
('presidenti', 0.5702304244041443)
('endors', 0.5668708086013794)
('#unpresid', 0.5659433603286743)
('teleprompt', 0.5657780766487122)

Query word: #allahsoil
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#teambt', 0.8091372847557068)
('#emirati', 0.8026494979858398)
('wil', 0.7765701413154602)
('#teamsuperjunior', 0.7684555053710938)
('islamophob', 0.768137514591217)
('occur', 0.7671984434127808)
('unifi', 0.7620072960853577)
('marx', 0.7586297392845154)
('correl', 0.7548872232437134)
('sweep', 0.7538445591926575)

Query word: #polit
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#uselect', 0.847956657409668)
('#libtard', 0.8324136734008789)
('libtard', 0.8282618522644043)
('#sjw', 0.8241668939590454)
('hidden', 0.8108854293823242)
('fascism', 0.8069114089012146)
('#stun', 0.8052216172218323)
('#liber', 0.7860569357872009)
('#grandmoth', 0.7750985026359558)
('#class', 0.772994339466095)

Query word: polit
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('inuit', 0.7180823683738708)
('perceiv', 0.7178562879562378)
('#rahulgandhi', 0.7148155570030212)
('delud', 0.6880771517753601)
('resourc', 0.6776108741760254)
('principl', 0.6752811670303345)
```

```
('nobl', 0.671585202217102)
('#chandigarh', 0.6554805040359497)
('repress', 0.6524448394775391)
('foreign', 0.6512322425842285)

Query word: #liber
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('libtard', 0.909777045249939)
('#libtard', 0.9092115163803101)
('#sjw', 0.9019615054130554)
('#stun', 0.8483309745788574)
('#hospit', 0.8278363943099976)
('astound', 0.8209948539733887)
('#ukip', 0.8096026182174683)
('#labour', 0.8027881383895874)
('cuck', 0.7967039346694946)
('#nuascannan', 0.7962537407875061)

Query word: liber
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('deen', 0.8012189269065857)
('#bernieorbust', 0.7922018766403198)
('repugn', 0.7693274021148682)
('discredit', 0.7676483988761902)
('unjust', 0.766100287437439)
('prez', 0.7642163634300232)
('leftist', 0.754870593547821)
('globalist', 0.7517481446266174)
('#arrog', 0.7483720779418945)
('ill', 0.7438279390335083)

Query word: #sjw
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#libtard', 0.9728550314903259)
('libtard', 0.960279107093811)
('#stun', 0.9193922281265259)
('#hospit', 0.9192678332328796)
('#liber', 0.9019616842269897)
('#nuascannan', 0.8629670143127441)
('astound', 0.8421329259872437)
('#labour', 0.838475227355957)
('#polit', 0.8241666555404663)
('tuck', 0.8073104619979858)

Query word: #retweet
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#boricua', 0.8311769366264343)
('#myminecraftrobot', 0.8203843832015991)
('#hispan', 0.8092269897460938)
('#tampa', 0.79936683177948)
('stomp', 0.7738752961158752)
('#pueorico', 0.7586437463760376)
('#venusexchang', 0.7561153769493103)
('#hottweet', 0.7558208107948303)
('#bbw', 0.7527008652687073)
('#vscolov', 0.750116229057312)

Query word: retweet
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#nuascannan', 0.6842193603515625)
('sweeti', 0.6556684374809265)
('owe', 0.6527565121650696)
('reto', 0.6362044215202332)
('#reachout', 0.6203826665878296)
('#flightofalifetim', 0.6173915863037109)
('shameless', 0.6150768995285034)
('critiqu', 0.609901487827301)
('#bbw', 0.6095160841941833)
('#curvi', 0.6076496839523315)

Query word: #miami
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#southbeach', 0.8586605191230774)
('#miamibeach', 0.8579981923103333)
('#tampa', 0.8394667506217957)
('#boricua', 0.8391954898834229)
('#wynwood', 0.8348535299301147)
('#familyday', 0.8341350555419922)
('stomp', 0.8209991455078125)
('#hispan', 0.7831558585166931)
('#espa', 0.778702437877655)
('#refreshvalet', 0.7719716429710388)

Query word: miami
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#waltdisneyworld', 0.7237499952316284)
('#meatloaf', 0.6959115266799927)
('#eastsid', 0.6929003596305847)
('collaps', 0.6896900534629822)
('walt', 0.6792576313018799)
('#pleasedontrain', 0.6788974404335022)
('#peakyblind', 0.677638828754425)
('speechless', 0.6769471764564514)
('#stamford', 0.6749014258384705)
('bonker', 0.6723107099533081)

Query word: #black
Top 10 sim words in X_train tweets by cos sim (word, sim score):
('#hispan', 0.8301653265953064)
('#boricua', 0.7819858193397522)
```

8. Discuss: how does w2v calculate the similarities?

Answer:

Word2Vec calculates the similarity between two words **u** and **v** using ***cosine similarity,*** which is the cosine of the *angle* **phi** between the *learned word-embedding vector representations* of words **u, v**.

Formally:

$$\cos\phi = sim(u,v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$

As can be seen from this equation, cosine similarity has at least two notable advantages compared with, for example, the squared Euclidean distance similarity function, which is used by some investigators, viz.:

1. **sim(u,v)** is ***normalized*** by the Euclidean magnitudes of **u, v** (denominator in above equation), and so is not influenced by arbitrarily large values in word embeddings (which, for example, *may be completely unrelated to* ***semantic meaning***); and,
2. The numerator in the cos similarity function above is the ***inner product*** of **u & v**, and thus will ***increase especially as their word embedding vectors (i.e., vector element values) become more similar: in other words, it represents a far more meaningful measure of semantic relatedness*** (at least according to the given model's particular ***word embeddings***, which it *learned from the specific input training corpus)* than a similarity function like simple *squared Euclidean distance.*

9. Discuss: do you think Word2Vec is supervised or unsupervised ML technique?

Answer:

Word2Vec with Skip-grams involves ***supervised learning.*** This model (after Mikolov et al., 2013) was developed as a simple, more efficient way to ***learn good word embeddings*** compared to, for example, the older *neural language model* approach. In the **Skip-gram** model, a collection of **context-target word pairs**, which constitute the *training examples* is extracted from a corpus in the following way:

1. Randomly pick a word to be a ***context word***.
2. Then, randomly pick another word in the corpus which lies within +/- 5 words (for example) of the above context word; designate this near neighbor as the context word's **target word** (this forms one **context-target word pair**.)
3. Repeat (1-2) above to generate a collection of context-target word pairs.
4. Use the collection of pairs in (3) above as **training examples** in a ***supervised learning model to learn a word embedding matrix, E*** for your corpus vocabulary, such that ***E • o(c) → e(c),*** where:
    - o(c) = one-hot vector for context word;
    - e(c) = context word *embedding vector*
5. Apply e(c) as input to a multi-class softmax regression function where each output class k (with its own associated parameter vector theta(k)) represents a potential target word in the corpus.
6. Minimize the log-loss of the model's target word prediction (for a given context word) in order to ***learn good word embeddings***.

10. Engineer features.

    For each tweet, we calculate the average of embeddings (function `word_vector`) and then apply it to every tidy tweet in `X_train` (use function `tokens_to_array`). Both functions are provided, inspect the code and save the features in `w2v_train`.

```python
from gensim.models.keyedvectors import KeyedVectors

def word_vector(tokens:list, size:int, keyed_vec:KeyedVectors= w2v.wv):
    vec = np.zeros(size).reshape((1, size))
    count = 0
    for word in tokens:
        try:
            vec += keyed_vec[word].reshape((1, size))
            count += 1
        except KeyError:
            # handling the case where the token is not in vocabulary
            continue
    if count != 0:
        vec /= count
    return vec

def tokens_to_array(tokens:list, size:int, keyed_vec:KeyedVectors= w2v.wv):
    array = np.zeros((len(tokens), size))
    for i in range(len(tokens)):
        array[i,:] = word_vector(tokens.iloc[i], size, keyed_vec=keyed_vec)
    return array
```

```python
w2v_train = tokens_to_array(tokenized_tweet, size=MAX_FEATURES, keyed_vec=w2v.wv)
```

```python
assert w2v_train.shape == (X_train.shape[0], MAX_FEATURES)
```

11. Prepare the test data before modeling for each approach:

- extract features from `X_test` using the bag of words approach; use `bow_vectorizer`
- extract features from `X_test` using the tf-idf approach; use `tfidf_vectorizer`
- extract features from `X_test` using Word2Vec embeddings; you need to first tokenized the tidy tweets in `X_test`, then convert the tokens to array of shape `(X_test.shape[0], MAX_FEATURES)`.

```python
bow_test = bow_vectorizer.transform(X_test)
tfidf_test = tfidf_vectorizer.transform(X_test)

tokenized_tweet_test = X_test.apply(lambda x: x.split())
w2v_test = tokens_to_array(tokenized_tweet_test, size=MAX_FEATURES, keyed_vec=w2v.wv)
```

```python
assert bow_test.shape == tfidf_test.shape == w2v_test.shape == (X_test.shape[0], MAX_FEATURES)
```

Task 4. Naive Bayes classifiers

In this task, you will build a naive Bayes), classifiers to identify the hate speech tweets using different sets of features from the last task, and evaluate their performances.

In the era of deep learning, naive Bayes is useful due to its simplicity and reasonable performance, especially if there is not much training data available. A common interview question is "Why is naive Bayes naive?".

We will use multi-variate Bernoulli naive Bayes `BernoulliNB`; try other flavors of naive Bayes if time permits. Code is pretty straightforward.

1. Import `BernoulliNB` for modeling and `classification_report` for reporting performance.

```python
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import classification_report
```

2. Create an instance of `BernoulliNB` named `BNBmodel`.

   We can use it for all three feature sets.

```python
# YOUR CODE HERE
BNBmodel = BernoulliNB()
```

3. Train the multi-variate Bernoulli naive Bayes using bag of words features and print the performance report.

```python
# YOUR CODE HERE (train the model)
bow_BNB = BNBmodel.fit(bow_train, y_train)
```

```python
# YOUR CODE HERE (report)
bow_BNB_pred = bow_BNB.predict(bow_test)
print(classification_report(y_test, bow_BNB_pred, digits=6))
```

```
              precision    recall  f1-score   support

           0   0.960000  0.972275  0.966098      7430
           1   0.557940  0.463458  0.506329       561

    accuracy                       0.936554      7991
   macro avg   0.758970  0.717866  0.736214      7991
weighted avg   0.931774  0.936554  0.933821      7991
```

4. Similarly, train the model using tf-idf features and print the performance report.

   Is the performance expected? Why or why not?

```
# YOUR CODE HERE
tfidf_BNB = BNBmodel.fit(tfidf_train, y_train)
```

```
# YOUR CODE HERE
tfidf_BNB_pred = tfidf_BNB.predict(tfidf_test)
print(classification_report(y_test, tfidf_BNB_pred, digits=6))
```

```
              precision    recall  f1-score   support

           0   0.960000  0.972275  0.966098      7430
           1   0.557940  0.463458  0.506329       561

    accuracy                       0.936554      7991
   macro avg   0.758970  0.717866  0.736214      7991
weighted avg   0.931774  0.936554  0.933821      7991
```

5. Finally, train the model using Word2Vec embeddings and report the performance.

```
# YOUR CODE HERE
w2v_BNB = BNBmodel.fit(w2v_train, y_train)
w2v_BNB_pred = w2v_BNB.predict(w2v_test)
print(classification_report(y_test, w2v_BNB_pred, digits=6))
```

```
              precision    recall  f1-score   support

           0   0.983502  0.850471  0.912162      7430
           1   0.290549  0.811052  0.427833       561

    accuracy                       0.847704      7991
   macro avg   0.637026  0.830761  0.669997      7991
weighted avg   0.934854  0.847704  0.878160      7991
```

```
print(type(w2v_BNB_pred), w2v_BNB_pred.shape)
```

```
<class 'numpy.ndarray'> (7991,)
```

6. Discuss the differences in performace using tf-idf vs skim-gram embeddings.

(See Answer Addendum below)

7. Examine a few tweets where the model(s) failed. What other features would you include in the next iteration?

Answer (also see more detailed Answer Addendum below):

Let's take a look at some *incorrect predictions* that the ***w2v_BNB model*** made on tidy_tweet's in `X_test`, in particular, errors made on hate tweets:

- **Type I errors** (false positives)
- **Type II errors** (false negatives)

```
print(type(y_test), y_test.shape)
```

```
<class 'pandas.core.series.Series'> (7991,)
```

```
# create a df of Type I and II prediction errors made by w2v_BNB model on
# tweets in X_test (let's collect num_exs of each):
# initialize:
fp_list = []    # list to hold false positive tidy tweets
fp_olist = []   # list to hold false positive tweets (semi-tidy full text)
fp_idx = []     # list of fp tweet orig. df indices

fn_list = []    # list to hold false negative tiny tweets
fn_olist = []   # list to hold false negative tweets (semi-tidy full text)
fn_idx = []     # list of fn tweet orig. df indices

num_exs = 21    # num of examples of each error type desired

# get list of orig. df indices from X_test, y_test:
idx_list = [i for i,_ in X_test.items()]
idy_list = [j for j,_ in y_test.items()]

# verify that the df indices & their order in X_test & y_test match:
assert len([(m,n) for m,n in zip(idx_list,idy_list) if m != n]) == 0

# verify that # test tweets, test gt labels, & test predictions match:
assert X_test.shape[0] == y_test.shape[0] == w2v_BNB_pred.shape[0]

# build lists of false positive and false negative examples,
# accumulating num_exs of ea. by looping through X_test tweets;
# store both tidy_tweet and corresponding near-orig. semi-tidy tweet (for comparison):
for n, idx in enumerate(idx_list):
    # check if have enough examples:
    if len(fp_list) == len(fn_list) == num_exs:
        break
    # find incorrect model pred:
    if w2v_BNB_pred[n] != y_test[idx]:
        if y_test[idx] == 0:
            assert w2v_BNB_pred[n] == 1              # a false positive pred
            if len(fp_list) < num_exs:               # fp_list not yet full
                fp_list.append(X_test[idx])          # store fp tidy tweet
                fp_olist.append(
                    raw_semi_tidy['tidy_tweet'][idx]) # store fp semi-tidy tweet
                fp_idx.append(idx)                   # store orig. df index for fp tweet

        else:                                        # a false negative pred
            assert y_test[idx] == 1 and w2v_BNB_pred[n] == 0
            if len(fn_list) < num_exs:               # fn_list not yet full
                fn_list.append(X_test[idx])          # store fn tidy tweet
                fn_olist.append(
                    raw_semi_tidy['tidy_tweet'][idx]) # store fn semi-tidy tweet
                fn_idx.append(idx)                   # store orig. df index for fn tweet

# create df of fp and fn model prediction examples:
fp_test_labels = [0 for t in range(num_exs)]
fn_test_labels = [1 for t in range(num_exs)]
fp_pred_labels = [1 for p in range(num_exs)]
fn_pred_labels = [0 for p in range(num_exs)]
```

```python

fp_error_df = pd.DataFrame(list(zip(fp_idx, fp_test_labels,
                                    fp_pred_labels, fp_list, fp_olist)),
                           columns = ['orig idx', 'gt lbl', \
                                      'prd lbl', 'tidy tweet', 'semi-tidy tweet'])
fn_error_df = pd.DataFrame(list(zip(fn_idx, fn_test_labels,
                                    fn_pred_labels, fn_list, fn_olist)),
                           columns = ['orig idx', 'gt lbl', \
                                      'prd lbl', 'tidy tweet', 'semi-tidy tweet'])

# combine above df's:
error_df = pd.concat([fp_error_df, fn_error_df], keys=['F+', 'F—'],
                     names=['Err', 'row'])
display(error_df)
```

Err	row	orig idx	gt lbl	prd lbl	tidy tweet	semi-tidy tweet
F+	**0**	21614	0	1	mate address argument rather #troll #staup #rude	mate try addressing my argument rather than #trolling me #staups #rude
	1	3197	0	1	surpris read make betray #scifi #kindleunlimit	a surprising read to make you gt gt gt the betrayal of ka lt lt lt #scifi #kindleunlimited
	2	31522	0	1	world turn polit zoolog disast govern truli ever learn	it s sad our world is turning into a political zoological disaster will our governments truly ever learn
	3	2100	0	1	cannot imagin shoe local enforc first respond #orlando prayer	i cannot imagine being in the shoes of local law enforcement and first responders #orlando prayers to all
	4	17364	0	1	#rahul forcibl ask #punjabi admit #drugaddict	#rahul forcibly asking #punjabis to admit they are #drugaddicts
	5	25085	0	1	year enough han accus bein accessori murder peopl auschwitz guard	years is not enough hanning is accused of bein an accessory ta the murder of people in auschwitz an ss guard from
	6	3719	0	1	coward presid #anerica #orlandoshoot	what a cowardly president we have in #anerica #orlandoshooting
	7	10706	0	1	tell player might penalti sure	so the tells all the other players there might be a penalty but they re not sure
	8	11109	0	1	nigga burn know femal burn know	these niggas burning amp won t let you know amp these females burning amp won t let you know
	9	23106	0	1	prayer #orlando #prayingfororlando #loveconquersh	prayers for #orlando #prayingfororlando #loveconquershate
	10	7955	0	1	decad later sick among airmen hydrogen bomb accid york time #comeonusa	decades later sickness among airmen after a hydrogen bomb accident the new york times #comeonusa
	11	8602	0	1	call person never use word eat word #hypocrit	she called your bf a n l personally have never used that word it s sad that she did amp now she is eating her words #hypocrit
	12	24648	0	1	#god merci countri nigeria news#god merci	#god should hv mercy on us as a country nigeria news#god will cry ur mercy
	13	5859	0	1	seat anywher houston stadium semi argentina seat front midfield	pat you can buy seats anywhere in the houston stadium for the semi vs argentina there were seats front row at midfield
	14	12636	0	1	garbag human celebr violenc	you re a garbage human for celebrating violence
	15	26700	0	1	#purfl member korean girl group crescendo music debut singl	#purfles is a member s korean girl group under crescendo music they debuted w the single on
	16	11353	0	1	bitch stand nigga even anymor	i got bitches that can t stand me over a nigga ion even gaf about anymore
	17	29829	0	1	confirm islam wors aid	confirmed islamism worse than aids
	18	21427	0	1	run world negat better suppo corrupt	the running the world cup is not a negative better than suppoing the corruption that is the ioc amp
	19	24571	0	1	went #mcdonald morn staff came ask blogger said said tweet everywher	went to #mcdonalds this morning amp a staff came by amp asked me if i m a blogger i said yes he said he saw my tweets amp i m everywhere
	20	6343	0	1	slaughterhous stale protest feed peopl bacon tumor	this pig on its way da slaughterhouse was staled by protesters they are feeding people bacon with tumors ht
F—	**0**	15634	1	0	look like brother first wife found anoth document	this guy look like his brother is who s first wife found him w another man having sex documented on a #tv
	1	27726	1	0	#black feel like stomp listen #retweet #tampa #miami	are #black amp feel like the are stomping on you listen #retweet #tampa #miami
	2	25629	1	0		you me too
	3	18746	1	0	almost alway trust brazilian	i almost always trust brazilians
	4	17180	1	0	real time back #smackdown #unitedstateschampion #uschampion #sdlive great #illeg talk	wow was a real at times back in the day on #smackdown #unitedstateschampion #uschampion #sdlive great more #illegal talk
	5	25494	1	0	check polic zero time month cours discrep noth	i ve had my id checked by the police zero times in months of course the discrepancy has nothing to do i m a wh
	6	14565	1	0	team #new show #phillysuppophilli combat	own team up with #new #mc amp show what #phillysuppophilly amp combat
	7	21294	1	0	silenc betray innoc #activ #hunger #nofur #animalright #noyulin #eah #wildlif #vegan #lgbt	silence betrays the innocent #activism #hunger #nofur #animalrights #noyulin #eah #wildlife #vegan #lgbt
	8	6074	1	0	never content back watch other right trampl upon right could next #iqg	never be content to sit back and watch as others rights are trampled upon your rights could be next #iqg #
	9	30793	1	0	drunk ladi fall heater #pubfrict indian friend tri help thrown #houston #midtown	a drunk lady falls on a heater in #pubfriction amp an my indian friend is trying to help but he is being thrown out #houston #midtown
	10	14965	1	0	meet wilmington peacekeep walk beauti #cnv week action #peac #povey #climat	meet the wilmington peacekeepers they walk in beauty #cnv week of actions #peace #povey #climate
	11	31817	1	0	unappet scam women need throw shackl #salad #food #foodi	an unappetizing scam women we need to throw off the shackles of #salad #food #foodie
	12	24449	1	0	birmingham friend miss special lectur bonhoeff racism bonhoeff exp	birmingham friends don t miss this special lecture on bonhoeffer and racism from bonhoeffer expe
	13	26696	1	0	know #environment	what you should know about #environmental
	14	24715	1	0	come #powerhungrytraitor #hereticfound	coming from and #powerhungrytraitors the #hereticfoundation
	15	11243	1	0	#black feel like listen #retweet #tampa #miami #newyork	are you #black amp feel like the are on you listen #retweet #tampa #miami #newyork
	16	25229	1	0	outrag racism rother hope famili fail follow advic life guard caus	outrageous racism from rother dc hope the families sue failing to follow advice for a life guard amp causing the d
	17	8290	1	0	haha	haha i get it
	18	18555	1	0	ohhhh peopl realli think #interraci relationship #childrenofcolor	ohhhh so people really do think that because you are in an #interracial relationship amp or have #childrenofcolor you opt out of
	19	21691	1	0	essenti read someon follow	essential reading by someone you should follow
	20	19612	1	0	mean #teammichaelpalag #teamcanc	you mean this guy #teammichaelpalage #teamcancer

Answer Addendum: Observations on model performance, and considerations for improving predictions:

1. Text pre-processing:

- Although **Snowball stemmer** (aka Porter2 stemmer) is widely used in NLP text pre-processing, it employs a ***simplistic rule-based heuristic*** which truncates words to their *'root' without regard to* ***context***. In the above, we see examples of *overstemming*, where too much of a word is cut off.
- Given the above, ***lemmatization*** might be tried instead during pre-processing. It may yield some performance gains, since it *takes into account* ***context:*** *viz.*, it requires knowledge of:

- a word's ***part-of-speech;*** &
- language ***structure;***

Lemmatization tries to take a somewhat more nuanced approach of reducing words to their ***dictionary/canonical form***.

- Of note, both ***stemming & lemmatization*** were designed to shift prediction performance in the direction of ***recall***, trading off ***precision*** as a result. We see evidence of this especially with the ***Word2Vec-based model:*** It's ***recall*** for *hate tweets* is nearly twice that of either ***bag of words- or tfidf-based models***; but its ***precision*** drops by *nearly ½ (compared to the others) to a dismal low of 0.291!*

2. Data considerations:

a. Class imbalance:

- That the *Word2Vec-based model's* recall is 0.816--far better than its precision of only 0.291--is perhaps not so surprising given the ***significant class imbalance*** in our tweet dataset viz., the ratio of non-hate- to hate-tweets is ***> 13 : 1***. This could be ameliorated by some of the techniques we've discussed in the past, including for example:
  - reducing the number of non-hate tweets in the training set;
  - artificially *augmenting* the number of hate tweets by creating additional training examples using various 'plausible' combinations of words or phrases derived from actual hate tweets in `X_train`;
  - adding more 'real' hate tweets drawn from the essentially unlimited number of readily available tweets

b. Sample size:

- Since we are not using pre-trained models, in addition to addressing class imbalance, simply collecting much more training data overall would likely be helpful. In our data, for example, the 'cleaned' tweets in `X_train` contain a total of only 161,101 (non-unique) words, of which only 12,001 (again, non-unique) are derived from hate tweets. This seems to be quite ***a small corpus from which to learn our own word embeddings from scratch*** (in the case of the Word2Vec-based model) or to derive our own simplistic features for the bag of words and tfidf-based models.

3. Model limitations:

Perhaps the most significant impediment to good performance here, however, is the decisive limitation imposed by the unsophisticated models we've used. Neither bag of words nor tfidf-based models take into account ***word sequence***, a key (arguably the most significant) determinant of ***syntactic meaning*** in language. Although *Word2Vec* trains on *context-target* pairs (word neighbors within a certain distance of one another), in the end, it is ***not formally trying to learn a probability distribution of word sequences and e.g. predict 'target' from 'context';*** rather, it is *really focused on learning good word embeddings*, which, no matter how good, however, won't bring us much closer to *accurately inferring meaning from the ordered arrangements of words that constitute language.* ***In short, we need a sequence model to achieve significant performance gains here.***

Task 5. Bidirectional LSTM

In this task, you will build a bidirectional LSTM (BiLSTM) model to detect tweets identified as hate speech, and visualize the embedding layer using Tensorboard projector.

Why BiLSTM? LSTM, at its core, preserves information from inputs that has already passed through it using the hidden state. Unidirectional LSTM only preserves information of the past because the only inputs it has seen are from the past. BiLSTMs run inputs in both ways, one from past to future and one from future to past and show very good results as they can understand context better ref.

1. Tokenizing and padding.

   As LSTM expects every sentence to be of the same length, in addition to `Tokenizer` with a given number of vocabulary `VOCAB_SIZE`, we need to pad shorter tweets with 0s until the length is `MAX_LEN` and truncate longer tweets to be exact `MAX_LEN` long.

   Function `tokenize_pad_sequences` is provided except that you need to supply correct `num_words` and `filters`; do NOT filter `#`.

   We feed the processed `tidy_tweet` to `tokenize_pad_sequences`, but one can perform the preprocessing steps in `Tokenizer` and apply it directly on the raw tweets.

```
VOCAB_SIZE = 25000
MAX_LEN = 50
```

```
import tensorflow as tf
from tensorflow import keras

from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences

def tokenize_pad_sequences(text):
    '''
    tokenize the input text into sequences of integers and then
    pad each sequence to the same length
    '''
    # Text tokenization
    tokenizer = Tokenizer(
        num_words=VOCAB_SIZE,
        filters='!"$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
        lower=True, split=' ', oov_token='oov')
    tokenizer.fit_on_texts(text)
    # Transforms text to a sequence of integers
    X = tokenizer.texts_to_sequences(text)
    # Pad sequences to the same length
    X = pad_sequences(X, padding='post', maxlen=MAX_LEN)

    return X, tokenizer
```

```
print('Before Tokenization & Padding \n', raw['tidy_tweet'][0])
X, tokenizer = tokenize_pad_sequences(raw['tidy_tweet'])
print('After Tokenization & Padding \n', X[0])
y = raw['label'].values
```

```
Before Tokenization & Padding
 father dysfunct selfish drag kid dysfunct #run
After Tokenization & Padding
 [  14 6492 2391 1685  211 6492  462    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0]
```

2. Let's split `X` into training and testing datasets, save 25% for testing. Then split training dataset into training and validation datasets, with 20% for validation. Set both `random_state` to be 42. Stratify both splits.

```
print(type(raw['label']), raw['label'].shape, type(y), y.shape)
```

```
<class 'pandas.core.series.Series'> (31962,) <class 'numpy.ndarray'> (31962,)
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
    )
X_train, X_val, y_train, y_val = train_test_split(
    X train, y train, test size=0.20, random state=42, stratify=y train
```

```
print('Train Set ->', X_train.shape, y_train.shape)
print('Validation Set ->', X_val.shape, y_val.shape)
print('Test Set ->', X_test.shape, y_test.shape)
```

```
Train Set -> (19176, 50) (19176,)
Validation Set -> (4795, 50) (4795,)
Test Set -> (7991, 50) (7991,)
```

3. Now build a sequential model:
    - an embedding layer
    - a bidirectional LSTM with 32 units and set `return_sequences=True` in LSTM
    - a global average pooling operation for temporal data
    - a dropout layer with 20% rate
    - a dense layer of 32 units and set the activation function to be ReLu
    - a dense layer of 1 unit and set the proper activation function for classification

```
from keras.models import Sequential
from tensorflow.keras import layers

EMBEDDING_DIM = 16
model = keras.Sequential(
    [
        layers.Embedding(input_dim=VOCAB_SIZE, output_dim=EMBEDDING_DIM),
        layers.Bidirectional(layers.LSTM(32, return_sequences=True)),
        layers.GlobalAveragePooling1D(),
        layers.Dropout(0.2),
        layers.Dense(32, activation='relu', name='dense_32'),
        layers.Dense(1, activation='sigmoid', name='dense_1_sigmoid')
    ]
)
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, None, 16)          400000    
                                                                 
 bidirectional (Bidirectiona  (None, None, 64)         12544     
 l)                                                              
                                                                 
 global_average_pooling1d (G  (None, 64)               0         
 lobalAveragePooling1D)                                          
                                                                 
 dropout (Dropout)           (None, 64)                0         
                                                                 
 dense_32 (Dense)            (None, 32)                2080      
                                                                 
 dense_1_sigmoid (Dense)     (None, 1)                 33        
                                                                 
=================================================================
Total params: 414,657
Trainable params: 414,657
Non-trainable params: 0
_________________________________________________________________
```

4. Compile the model.

    Fill in a proper loss function and use adam as the optimizer. For metrics, include precision and recall in the metrics, in addition to accuracy.

```
from keras.metrics import Precision, Recall
model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['Precision', 'Recall', 'Accuracy']
    )
```

5. Train the model for 10 epochs on training dataset with a validation set.

```
EPOCHS=10
BATCH_SIZE = 32
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    batch_size=BATCH_SIZE, epochs=EPOCHS, verbose=2)
```

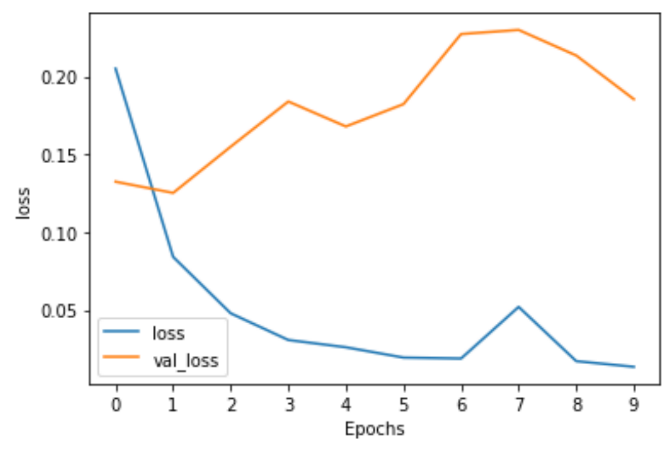
```
Epoch 1/10
600/600 - 17s - loss: 0.2051 - precision: 0.7025 - recall: 0.2528 - Accuracy: 0.9401 - val_loss: 0.1324 - val_precision: 0.7731 - val_recall: 0.5476 - val_Accuracy: 0.9570 - 17s/epoch - 28ms/step
Epoch 2/10
600/600 - 5s - loss: 0.0841 - precision: 0.8605 - recall: 0.7428 - Accuracy: 0.9735 - val_loss: 0.1252 - val_precision: 0.8205 - val_recall: 0.5714 - val_Accuracy: 0.9612 - 5s/epoch - 9ms/step
Epoch 3/10
600/600 - 5s - loss: 0.0478 - precision: 0.9209 - recall: 0.8654 - Accuracy: 0.9853 - val_loss: 0.1550 - val_precision: 0.7704 - val_recall: 0.6190 - val_Accuracy: 0.9604 - 5s/epoch - 9ms/step
Epoch 4/10
600/600 - 5s - loss: 0.0307 - precision: 0.9418 - recall: 0.9264 - Accuracy: 0.9908 - val_loss: 0.1840 - val_precision: 0.7032 - val_recall: 0.6488 - val_Accuracy: 0.9562 - 5s/epoch - 9ms/step
Epoch 5/10
600/600 - 5s - loss: 0.0260 - precision: 0.9538 - recall: 0.9361 - Accuracy: 0.9923 - val_loss: 0.1679 - val_precision: 0.8276 - val_recall: 0.5714 - val_Accuracy: 0.9616 - 5s/epoch - 9ms/step
Epoch 6/10
600/600 - 5s - loss: 0.0194 - precision: 0.9729 - recall: 0.9606 - Accuracy: 0.9954 - val_loss: 0.1823 - val_precision: 0.7232 - val_recall: 0.6220 - val_Accuracy: 0.9568 - 5s/epoch - 9ms/step
Epoch 7/10
600/600 - 5s - loss: 0.0188 - precision: 0.9699 - recall: 0.9599 - Accuracy: 0.9951 - val_loss: 0.2274 - val_precision: 0.6189 - val_recall: 0.6429 - val_Accuracy: 0.9472 - 5s/epoch - 9ms/step
Epoch 8/10
600/600 - 5s - loss: 0.0520 - precision: 0.8518 - recall: 0.8974 - Accuracy: 0.9819 - val_loss: 0.2300 - val_precision: 0.6625 - val_recall: 0.6369 - val_Accuracy: 0.9518 - 5s/epoch - 9ms/step
Epoch 9/10
600/600 - 5s - loss: 0.0171 - precision: 0.9730 - recall: 0.9628 - Accuracy: 0.9955 - val_loss: 0.2136 - val_precision: 0.7114 - val_recall: 0.6310 - val_Accuracy: 0.9562 - 5s/epoch - 9ms/step
Epoch 10/10
600/600 - 5s - loss: 0.0135 - precision: 0.9784 - recall: 0.9755 - Accuracy: 0.9968 - val_loss: 0.1855 - val_precision: 0.7500 - val_recall: 0.6250 - val_Accuracy: 0.9591 - 5s/epoch - 9ms/step
```

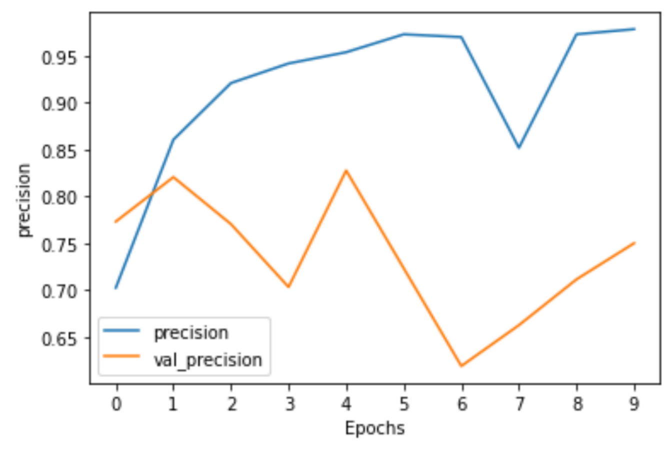
6. Function `plot_graphs` is provided below to visualize how the performance of model progresses as a function of epoch.

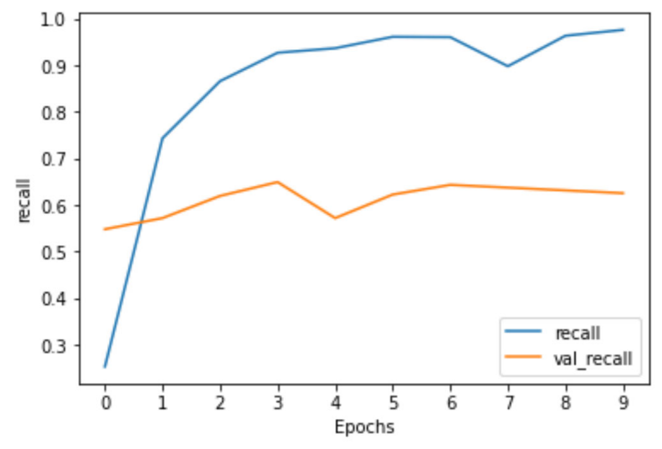
    Visualize accuracy and loss.

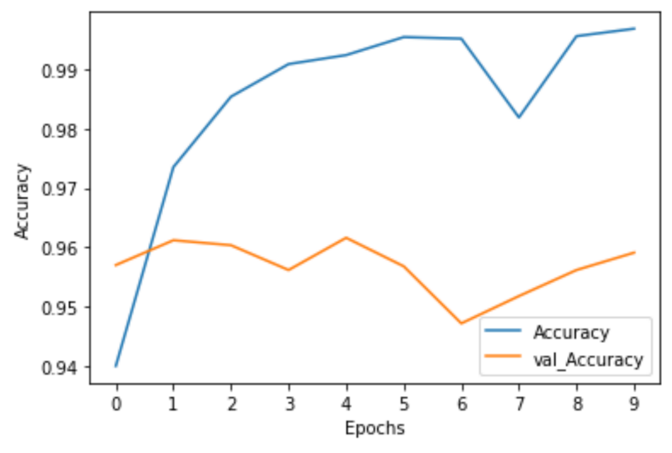
```
def plot_graphs(history, metric):
  fig, ax = plt.subplots()
  plt.plot(history.history[metric])
  plt.plot(history.history['val_'+metric], '')
  ax.set_xticks(range(EPOCHS))
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])
```

```
# YOUR CODE HERE
plot_graphs(history,'loss')
plot_graphs(history,'precision')
plot_graphs(history,'recall')
plot_graphs(history,'Accuracy')
```









7. The model starts to overfit after a couple of epochs. Consider using early stopping to stop training when a monitored metric has stopped improving.

   What can we do to tame overfitting?

Answer:

Measures that we can use to mitigate the above overfitting include:

- **Early stopping** *(as suggested):* While the training loss continues to decrease monotonically over epochs, the **validation loss** starts to progressively increase from about *epoch 2*:
  - Where to stop depends in large part on our **goal:** (1) Do we want to make *absolutely sure* that we don't miss *any* hate tweets? (⇒ maximize recall with the constraint that precision be still acceptable); or, (2) Do we want to make sure we don't censure *any* non-hate tweets? (⇒ maximize precision within the bounds of acceptable recall).
  - In our case above, how to **trade off recall vs. precision** is made somewhat more straightforward by the fact that recall in the validation set doesn't much change from about epoch 2 onwards (≈ 0.6). On the other hand, validation **precision** dips early in training (at epoch 1), nearly completely recovers in epoch 2, but then declines over subsequent epochs (with a partial recovery at epoch 4). Although there is a slight drop in validation recall at epoch 2, validation loss has not yet significantly diverged at this point. Thus, epoch 2 may represent a reasonable tradeoff (again, depending on our specific goals). *NB: Given random batches, these curves can vary depending on the specific run (for instance, on a previous run, epoch 4 looked like a reasonable recall vs. precision tradeoff point for early stopping).*
- **Get more data:** A larger, more diverse dataset, which also has less class imbalance--as long as it is generally representative of the data we intend to run inference on--will help *reduce overfitting.*
- **Consider adjusting the number of hidden units:** Although the number of hidden units shown in the model summary above seems reasonable compared to the dimension of the embedding space, reducing this value may result in less overfitting.
- **Consider adjusting the embedding space:** The embedding-space is only 16-dimensional, which may not capture enough of the nuances of meaning of words in our tweet dataset. Increasing the length of our word embedding representations might help with the observed overfitting.
- **Leverage a pre-trained model:** In particular, we could select a HuggingFace model trained on a large corpus of tweets.

8. Print the classification report of the model on test dataset.

```
# YOUR CODE HERE
# Looking at keras metrics on X_test using model.evaluate with batch_size=32:
print('EVALUATE model on X_test with batch_size = 32:\n')
results = model.evaluate(X_test, y_test, batch_size=32)

print(f'\nModel test loss, precision, recall, & Accuracy, respectively,\n'
      f'after {EPOCHS:2d} epochs of model training with training \n'
      f'batch_size of {BATCH_SIZE:2d} and model.evaluate batch_size of {BATCH_SIZE:2d}: \n'
      f'{results}\n')

# Looking at keras metrics on X_test using model.evaluate with batch_size=1:
print('EVALUATE model on X_test with batch_size = 1, i.e., all in one go:\n')
results = model.evaluate(X_test, y_test, batch_size=1)

print(f'\nModel test loss, precision, recall, & Accuracy, respectively,\n'
      f'after {EPOCHS:2d} epochs of model training with training \n'
      f'batch_size of {BATCH_SIZE:2d} and model.evaluate batch_size of ONE: \n'
      f'{results}\n')

# classification report:
print('\nClassification report on keras model.predict:\n')
y_pred = model.predict(X_test) > 0.5
print(classification_report(y_test, y_pred, digits=6))
```

```
EVALUATE model on X_test with batch_size = 32:

250/250 [==============================] - 1s 5ms/step - loss: 0.1934 - precision: 0.6911 - recall: 0.6221 - Accuracy: 0.9539

Model test loss, precision, recall, & Accuracy, respectively,
after 10 epochs of model training with training
batch_size of 32 and model.evaluate batch_size of 32:
[0.1934225857257843, 0.6910890936851501, 0.6221033930778503, 0.9539481997489929]

EVALUATE model on X_test with batch_size = 1, i.e., all in one go:

7991/7991 [==============================] - 37s 5ms/step - loss: 0.1934 - precision: 0.6911 - recall: 0.6221 - Accuracy: 0.9539

Model test loss, precision, recall, & Accuracy, respectively,
after 10 epochs of model training with training
batch_size of 32 and model.evaluate batch_size of ONE:
[0.19342517852783203, 0.6910890936851501, 0.6221033930778503, 0.9539481997489929]


Classification report on keras model.predict:

250/250 [==============================] - 2s 3ms/step
              precision    recall  f1-score   support

           0   0.971680  0.979004  0.975329      7430
           1   0.691089  0.622103  0.654784       561

    accuracy                       0.953948      7991
   macro avg   0.831385  0.800554  0.815056      7991
weighted avg   0.951982  0.953948  0.952825      7991
```

9. Discuss: how does the BiLSTM model improve the classification over naive Bayes?

Answer:

The BiLSTM model resulted in significant overall improvement in performance compared to our previous models. This certainly is in line with our expectations, since as mentioned, the BiLSTM model is the only one which incorporated **sequence information**, *and it does so in both directions(!)*. Sequence and context (both forward 'into the future', and backward 'looking into the past') are critical aspects of inferring meaning in language.

As the the classification reports show, the BiLSTM improved **both recall (0.48 in tf-idf) to 0.62 *as well as* precision (0.55 in tf-idf) to 0.69**. Although the word2vec model boosted recall *to the highest observed level in this notebook (0.82), it did so at the expense of a significant drop in precision (0.29!), and exhibited the lowest overall f1-score = 0.43* (with **f1 being the key performance metric here given the significant class imbalance)**. Overall, ***the BiLSTM gave the best f1-score of 0.655*** for hate-tweet prediction.

```
# # NB using tf-idf
#               precision    recall  f1-score   support

#            0       0.96      0.97      0.97      7430
#            1       0.55      0.48      0.51       561

#     accuracy                           0.94      7991
#    macro avg       0.75      0.72      0.74      7991
# weighted avg       0.93      0.94      0.93      7991

# # NB using word2vec
#               precision    recall  f1-score   support

#            0       0.98      0.85      0.91      7430
#            1       0.29      0.82      0.43       561

#     accuracy                           0.85      7991
#    macro avg       0.64      0.83      0.67      7991
# weighted avg       0.94      0.85      0.88      7991
```

10. Visualize embeddings using Embedding Projector in Tensorboard. The setup for Tensorboard can be tricky, most of the code is provided.

    TensorBoard reads tensors and metadata from the logs of your tensorflow projects. The path to the log directory is specified with log_dir below.

    In order to load the data into Tensorboard, we need to save a training checkpoint to that directory, along with metadata that allows for visualization of a specific layer of interest in the model.

    Load the TensorBoard notebook extension and import `projector` from `tensorboard.plugins`.

```
%load_ext tensorboard
```

```
from tensorboard.plugins import projector
```

11. Clear any logs from previous runs if any.

```
rm -rf /logs/
```

12. Set up a logs directory, so Tensorboard knows where to look for data.

```
import os
log_dir='/logs/tweets-example/'
if not os.path.exists(log_dir):
    os.makedirs(log_dir)
```

13. Save the first `VOCAB_SIZE` most frequent words in the vocabulary as `metadata.tsv`.

```
with open(os.path.join(log_dir, 'metadata.tsv'), "w") as f:
  i = 0
  for label in tokenizer.word_index.keys():
    if label == 'oov':
      continue # skip oov
    f.write("{}\n".format(label))
    if i > VOCAB_SIZE:
      break
    i += 1
```

14. Save the weights we want to analyze as a variable. Note that the first value represents any unknown word, which is not in the metadata, here we will remove this value.

```
weights = tf.Variable(model.layers[0].get_weights()[0][1:]) # `embeddings` has a shape of (num_vocab, embedding_dim)
```

15. Create a checkpoint from embedding, the filename and key are the name of the tensor.

```
checkpoint = tf.train.Checkpoint(embedding=weights)
checkpoint.save(os.path.join(log_dir, "embedding.ckpt"))
```

```
'/logs/tweets-example/embedding.ckpt-1'
```

16. Set up config.

```
config = projector.ProjectorConfig()
embedding = config.embeddings.add()
```

17. The name of the tensor will be suffixed by `/.ATTRIBUTES/VARIABLE_VALUE`.

```
embedding.tensor_name = "embedding/.ATTRIBUTES/VARIABLE_VALUE"
embedding.metadata_path = 'metadata.tsv'
projector.visualize_embeddings(log_dir, config)
```

18. Verify the following files exist under the current directory

```
ls /logs/tweets-example/
```

```
checkpoint                            metadata.tsv
embedding.ckpt-1.data-00000-of-00001  projector_config.pbtxt
embedding.ckpt-1.index
```

19. Now run Tensorboard against on log data we just saved.

You may need to run this cell **twice** to see the projector correctly. Use Chrome for least friction.

```
%tensorboard --logdir /logs/tweets-example/
```

TensorBoard INACTIVE

No dashboards are active for the current data set.

Probable causes:

- You haven't written any data to your event files.
- TensorBoard can't find your event files.

If you're new to using TensorBoard, and want to find out how to add data and set up your event files, check out the README and perhaps the TensorBoard tutorial.

If you think TensorBoard is configured properly, please see the section of the README devoted to missing data problems and consider filing an issue on GitHub.

Last reload: Nov 9, 2022, 5:03:48 PM

Log directory: /logs/tweets-example/

The TensorBoard Projector can be a great tool for interpreting and visualzing embedding. The dashboard allows users to search for specific terms, and highlights words that are adjacent to each other in the embedding (low-dimensional) space. Try a few word in the Search box and see if the embeddings make sense.

Task 6. Interpretation

Lastly let's try to understnad predictions by BiLSTM using a model agnostic approach -- Local interpretable model-agnostic explanations (LIME)

1. Import `LimeTextExplainer` from the `lime_text` module in package `lime`

```
from lime.lime_text import LimeTextExplainer
```

2. Create an instance of `LimeTextExplainer`, call it `explanier`.

```
explainer = LimeTextExplainer(class_names=['no', 'yes'], random_state=2)
```

3. Method `explain_instance` expects the `classifier_fn` to be a function, we provide the function `predict_proba` as below.

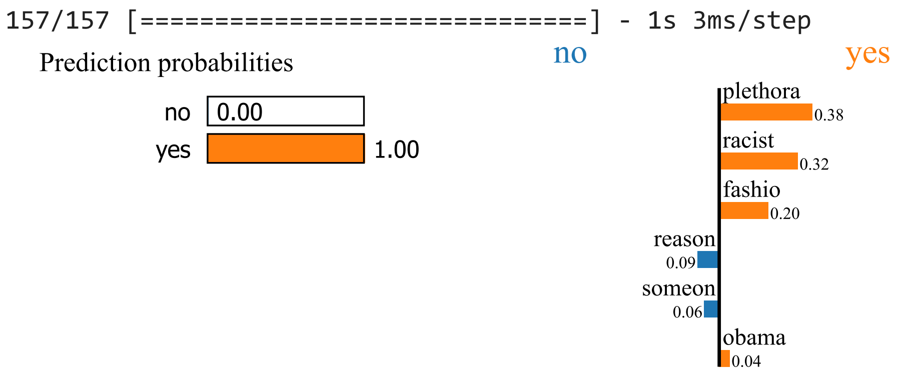
```
def predict_proba(arr):
    processed = tokenizer.texts_to_sequences(arr)
    processed = pad_sequences(processed, padding='post', maxlen=MAX_LEN)
    pred = model.predict(processed)
    r = []
    for i in pred:
        temp = i[0]
        r.append(np.array([1-temp,temp]))
    return np.array(r)
```

4. Read about `explain_instance`.

   Create an instance named `exp` to explain the 16399th tidy tweet from the original dataset, i.e., `raw.tidy_tweet.iloc[16399]`.

```
idx = 16399
exp = explainer.explain_instance(
    raw.tidy_tweet.iloc[idx],
    classifier_fn=predict_proba,
    num_features=6)
exp.show_in_notebook(text=raw.tidy_tweet.iloc[idx])
```

```
157/157 [==============================] - 1s 3ms/step
```



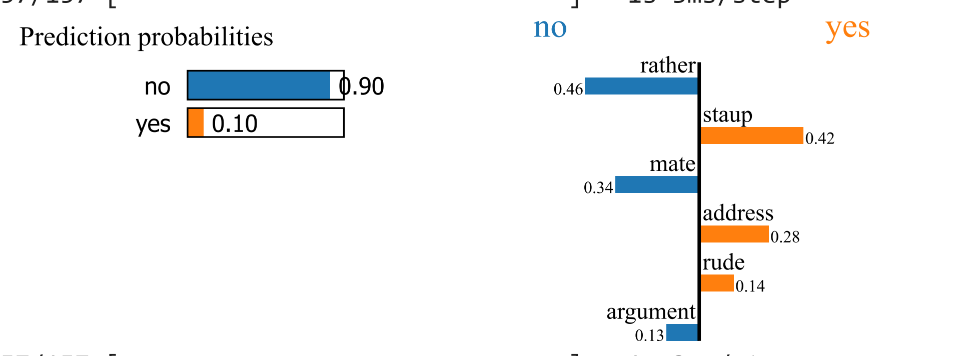
Text with highlighted words

america racist plethora reason someon like obama ignor mani fashio

5. Pick another random tweet and generate explanations for the prediction.

```
# YOUR CODE HERE

# Instead of generating explanations for random tweet predictions, let's look
# at explanations for the 21 examples of false pos & false neg tweet predictions
# made by the less 'performant' Word2Vec-based naive Bayes classifier used
# earlier:

# function to create explain_instance for a given tidy_tweet:
def exp_lime(false_tweet):
    exp = explainer.explain_instance(
        false_tweet,
        classifier_fn=predict_proba,
        num_features=6)
    exp.show_in_notebook(text=false_tweet)

# 'explain' false pos/neg predictions by the less performant Word2Vec-based model
# using LIME (21 false pos Word2Vec predictions displayed first, followed by
# 21 false neg preds):
# (explainer seems to have a problem with item #24; let's try to skip over it)
for i, fttw in enumerate(error_df['tidy tweet']):
    if i == 23:
        continue
    exp_lime(fttw)
```

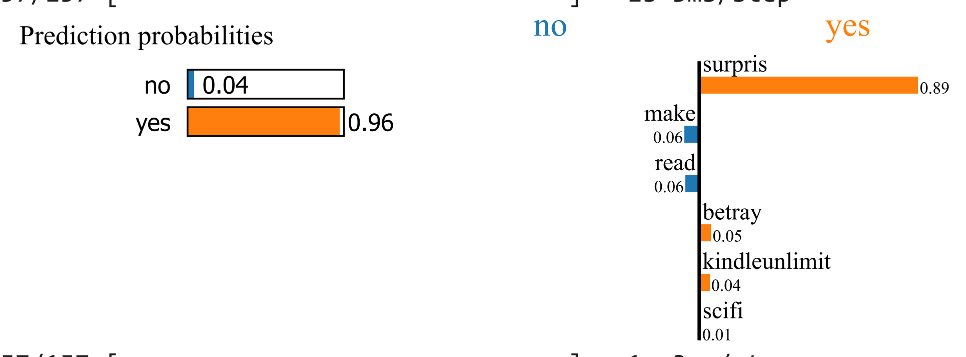
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

mate address argument rather #troll #staup #rude

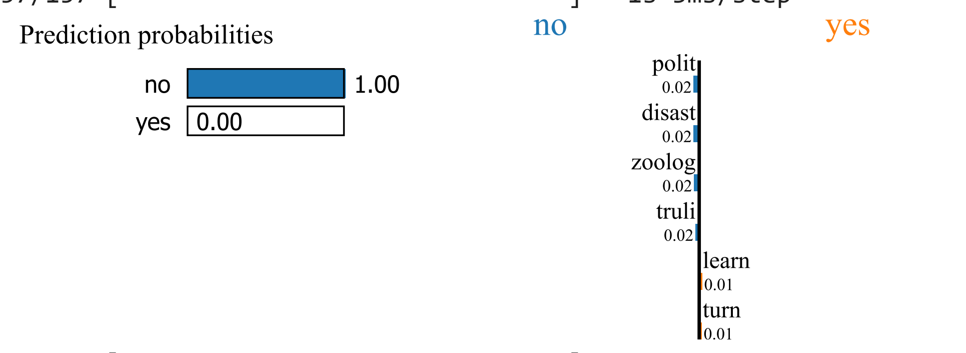
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

surpris read make betray #scifi #kindleunlimit

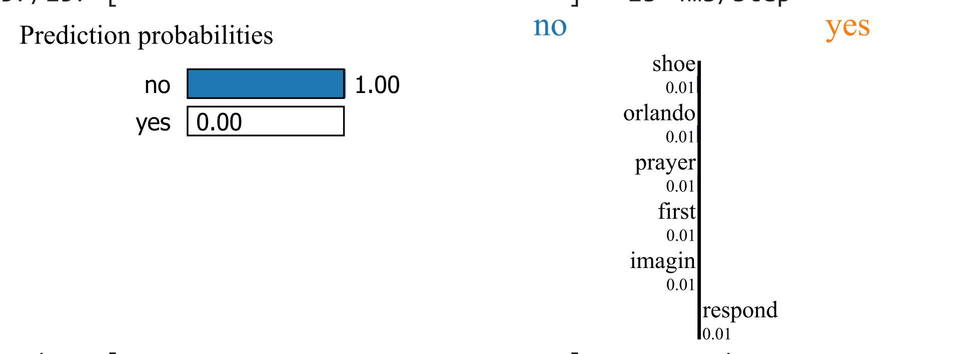
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

world turn polit zoolog disast govern truli ever learn

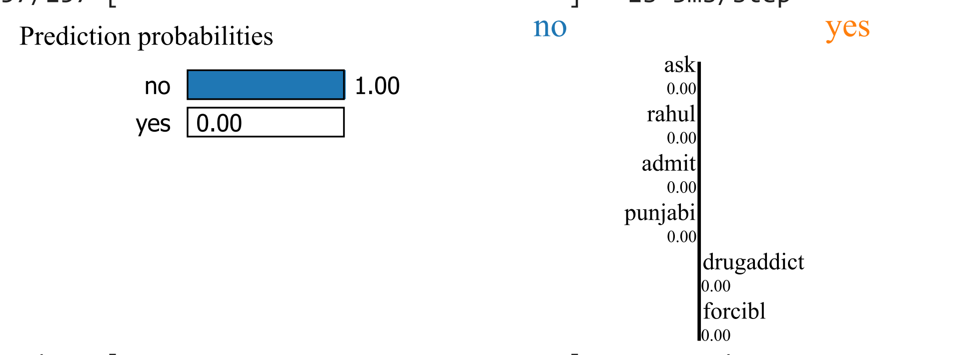
```
157/157 [==============================] - 1s 4ms/step
```



Text with highlighted words

cannot imagin shoe local enforc first respond #orlando prayer

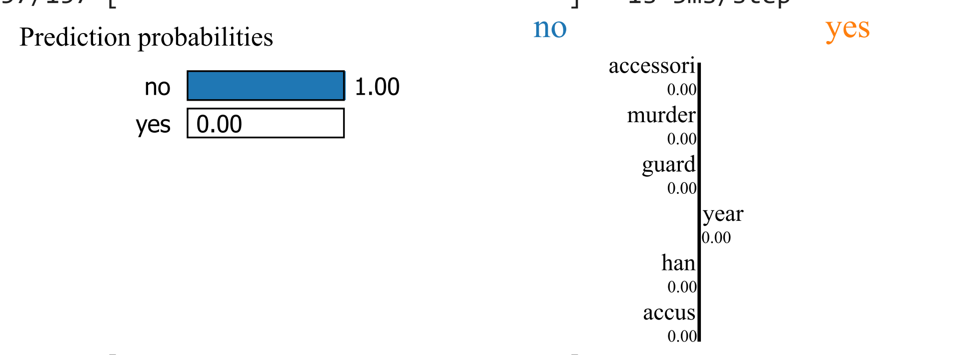
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

#rahul forcibl ask #punjabi admit #drugaddict

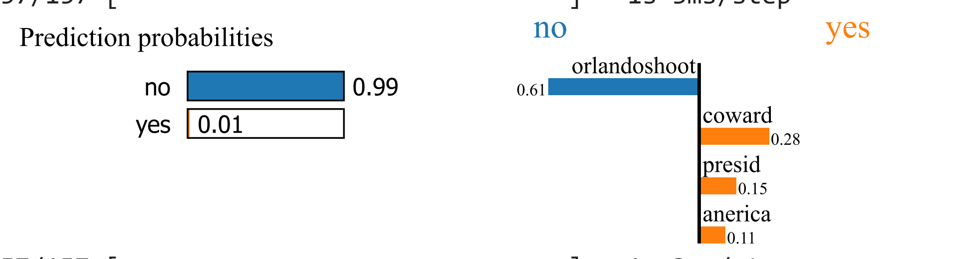
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

year enough han accus bein accessori murder peopl auschwitz guard

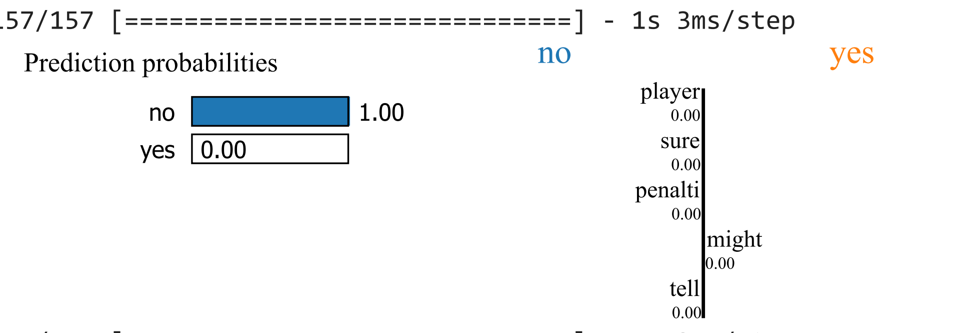
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

coward presid #anerica #orlandoshoot

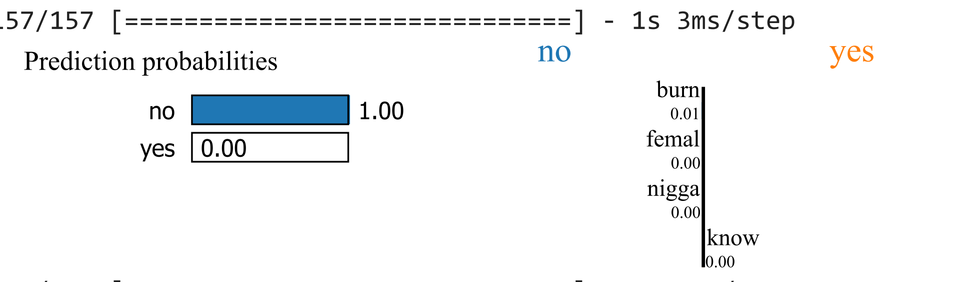
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

tell player might penalti sure

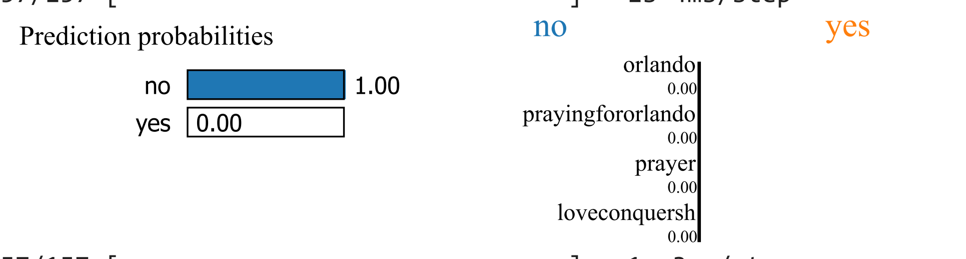
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

nigga burn know femal burn know

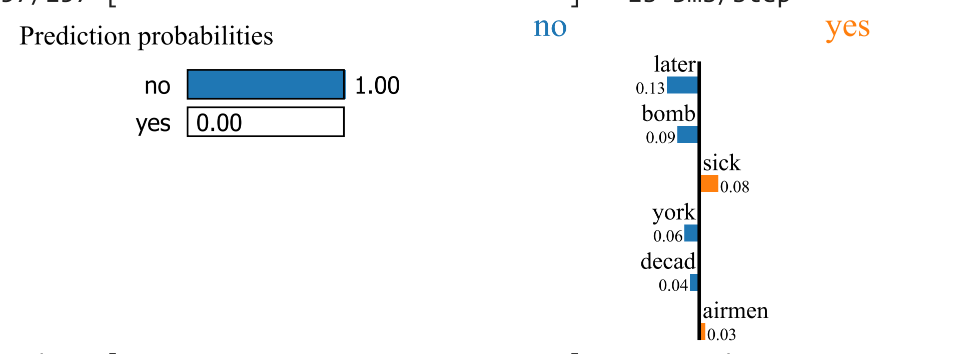
```
157/157 [==============================] - 1s 4ms/step
```



Text with highlighted words

prayer #orlando #prayingfororlando #loveconquersh

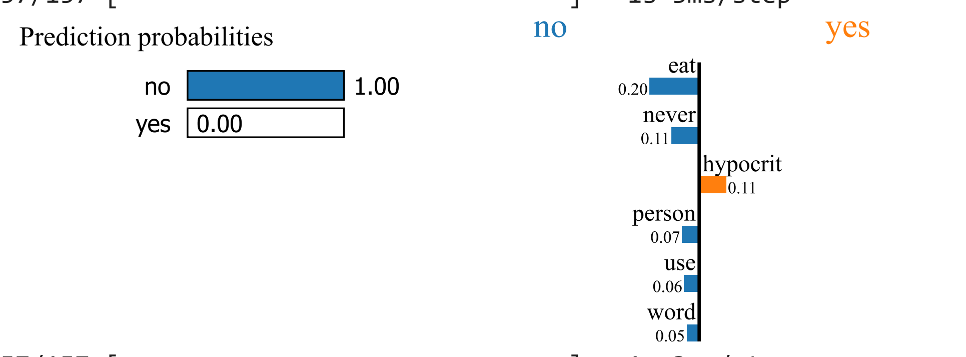
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

decad later sick among airmen hydrogen bomb accid york time #comeonusa

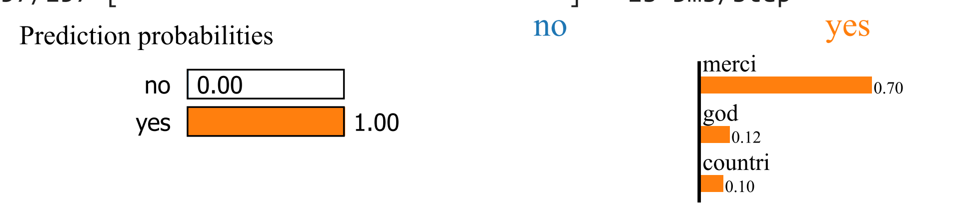
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

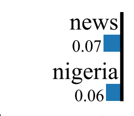
call person never use word eat word #hypocrit

```
157/157 [==============================] - 1s 3ms/step
```

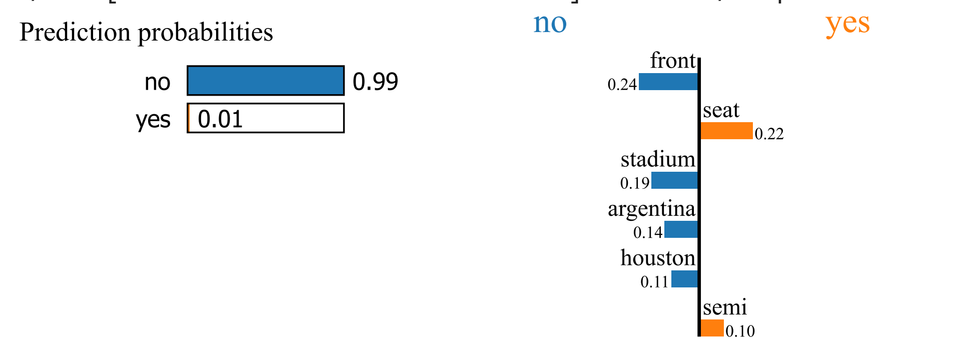


Text with highlighted words

#god merci countri nigeria news#god merci



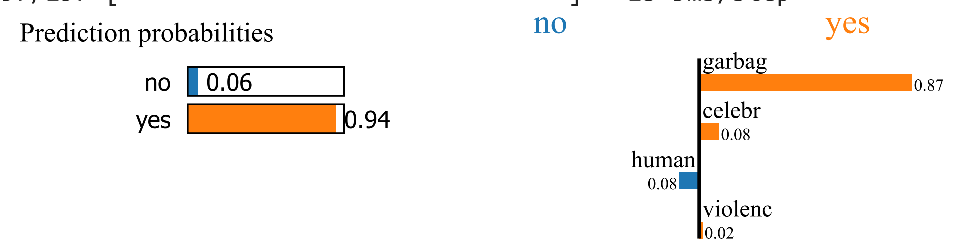
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

seat anywher houston stadium semi argentina seat front midfield

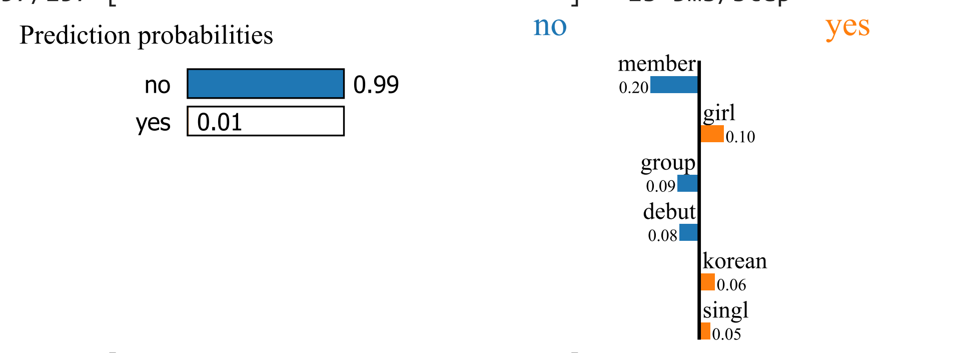
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

garbag human celebr violenc

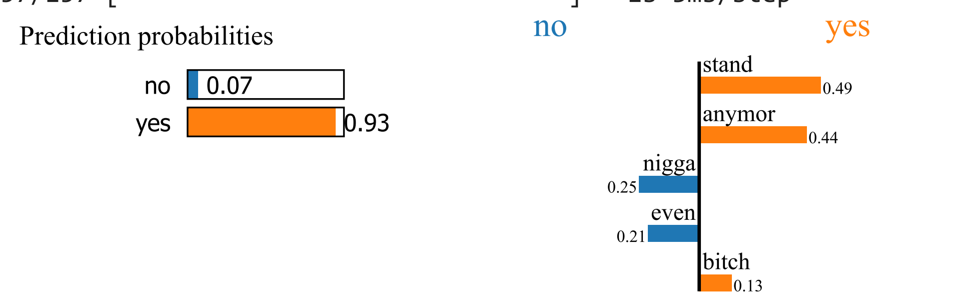
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

#purfl member korean girl group crescendo music debut singl

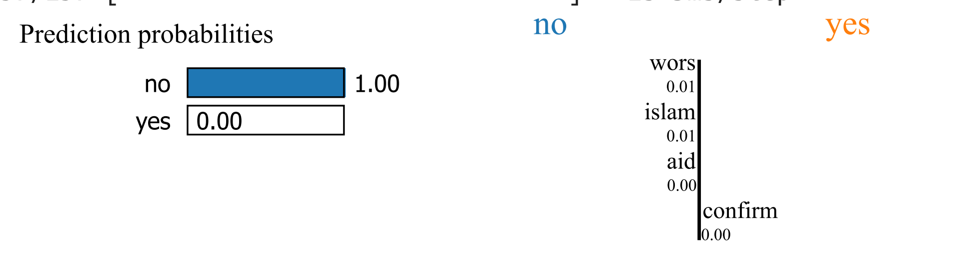
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

bitch stand nigga even anymor

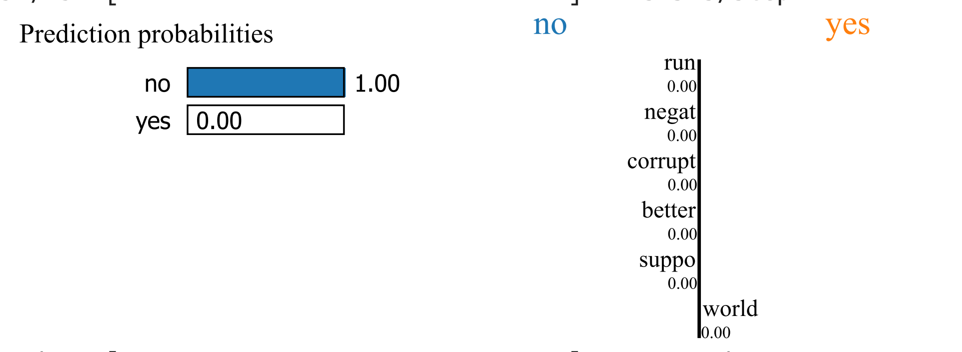
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

confirm islam wors aid

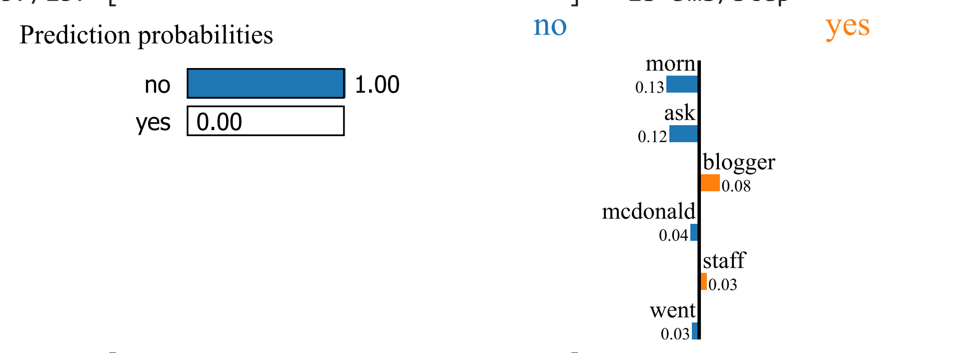
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

run world negat better suppo corrupt

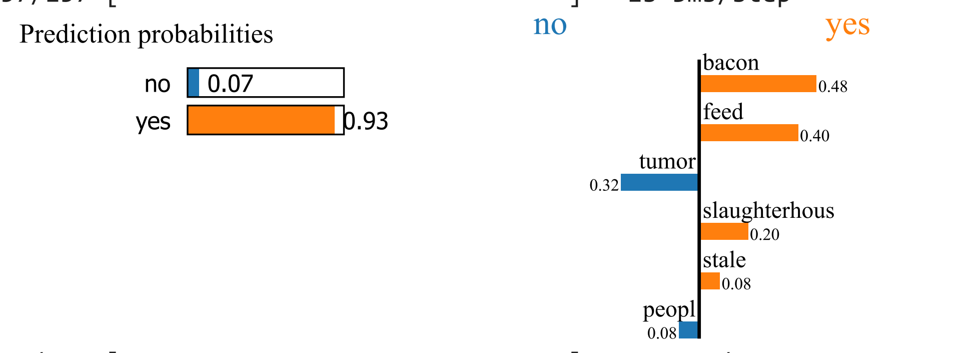
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

went #mcdonald morn staff came ask blogger said said tweet everywher

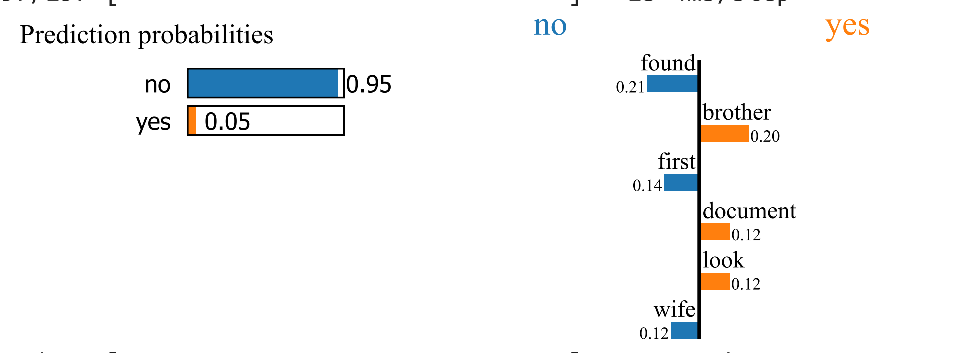
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

slaughterhous stale protest feed peopl bacon tumor

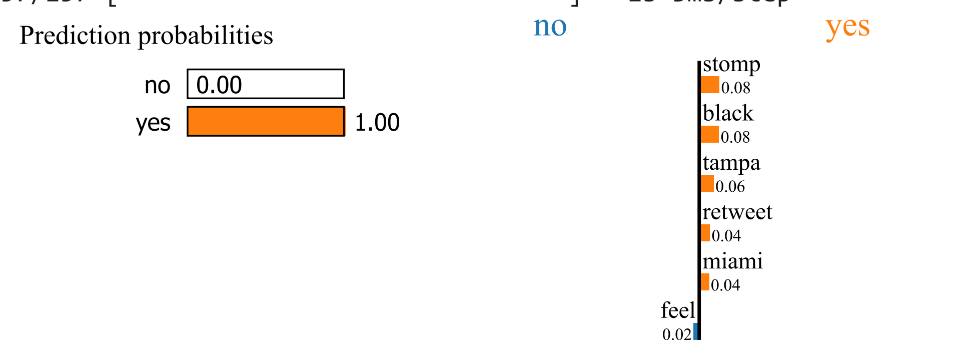
```
157/157 [==============================] - 1s 4ms/step
```



Text with highlighted words

look like brother first wife found anoth document

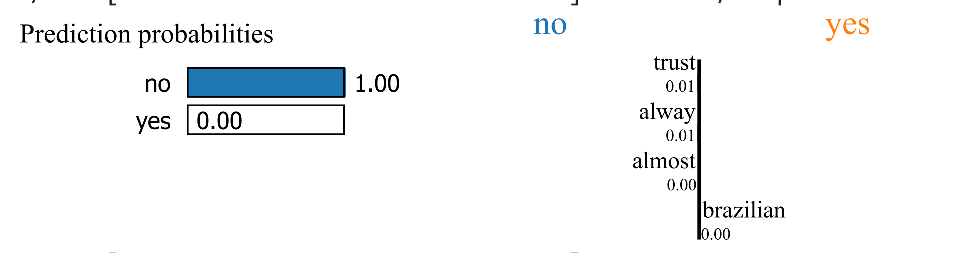
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

#black feel like stomp listen #retweet #tampa #miami

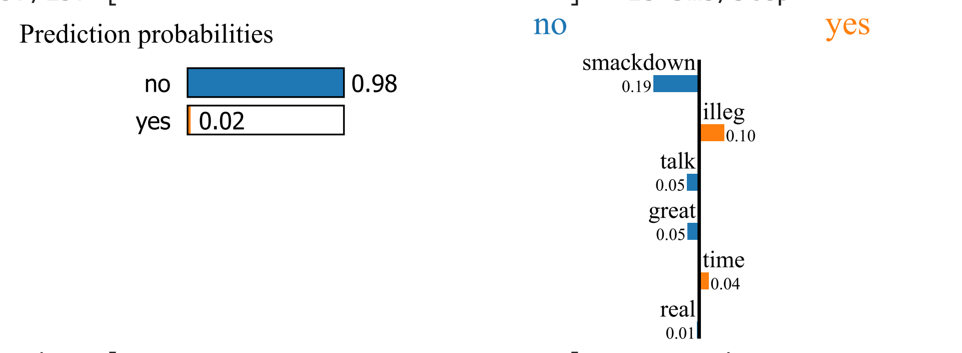
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

almost alway trust brazilian

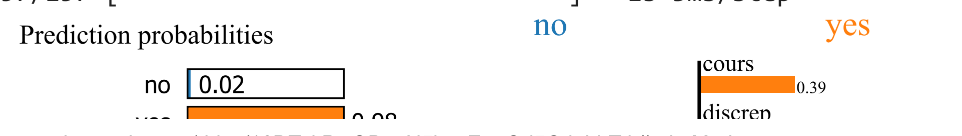
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

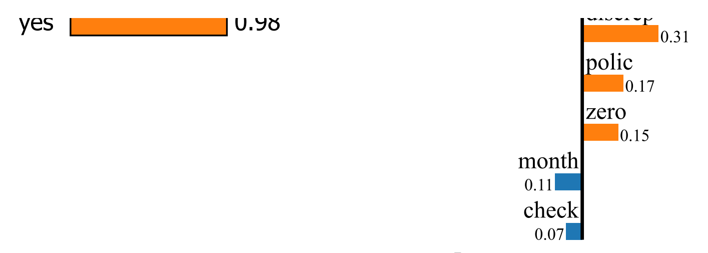
real time back #smackdown #unitedstateschampion #uschampion #sdlive great #illeg talk

```
157/157 [==============================] - 1s 3ms/step
```

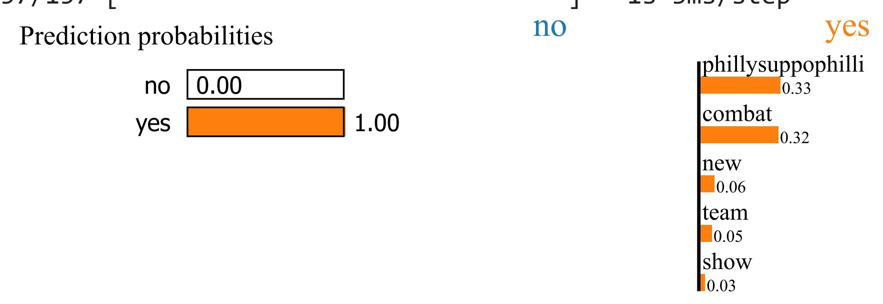


Text with highlighted words

check polic zero time month cours discrep noth



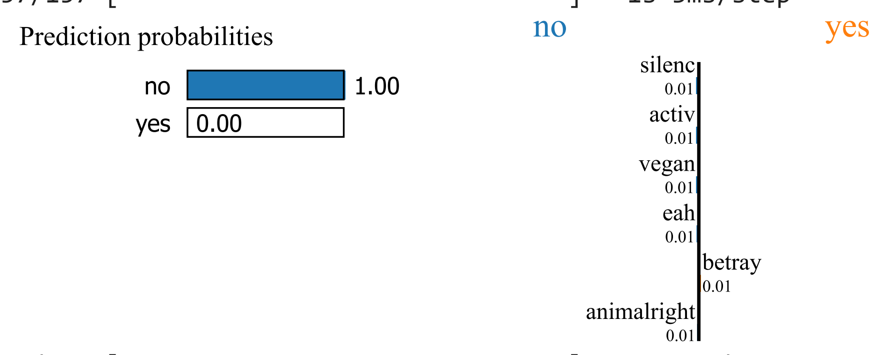
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

team #new show #phillysuppophilli combat

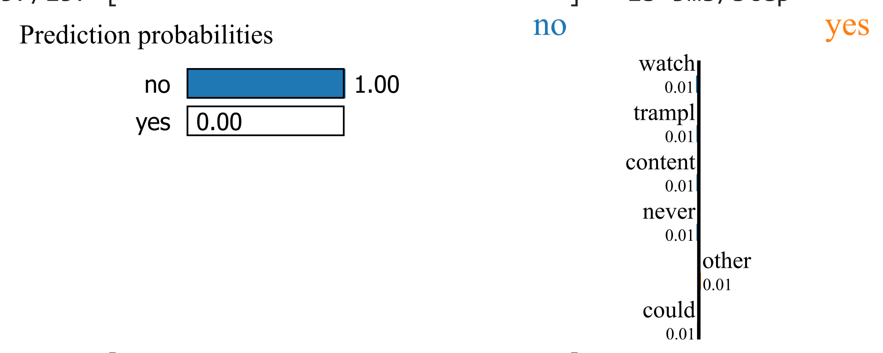
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

silenc betray innoc #activ #hunger #nofur #animalright #noyulin #eah #wildlif #vegan #lgbt

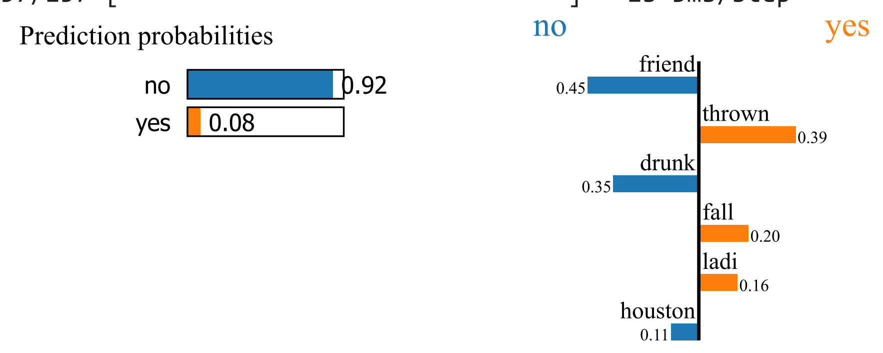
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

never content back watch other right trampl upon right could next #iqg

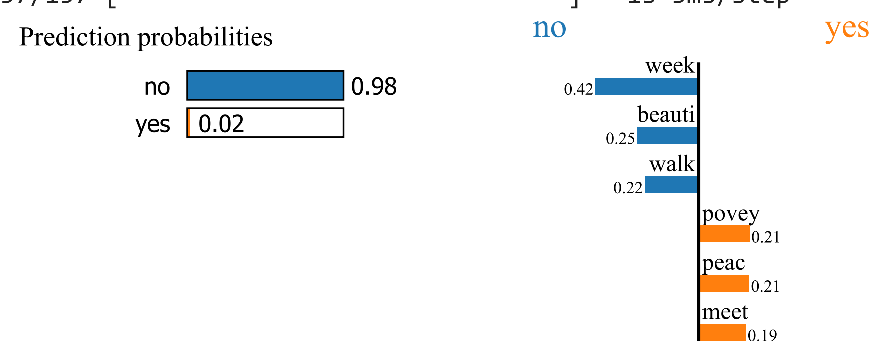
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

drunk ladi fall heater #pubfrict indian friend tri help thrown #houston #midtown

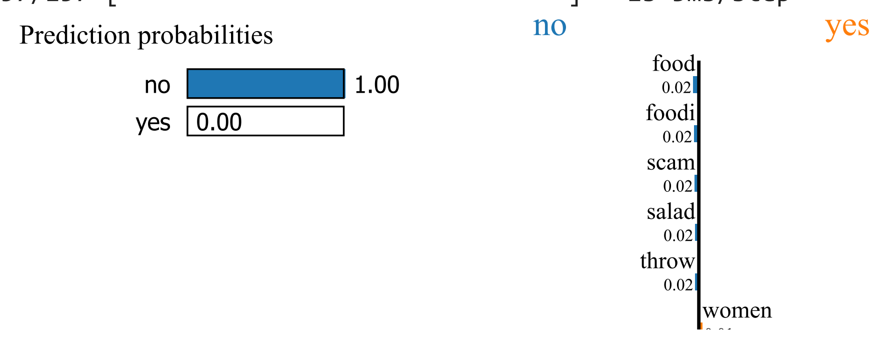
```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

meet wilmington peacekeep walk beauti #cnv week action #peac #povey #climat

```
157/157 [==============================] - 1s 3ms/step
```



Text with highlighted words

unappet scam women need throw shackl #salad #food #foodi

6. Jot down your observations in explaining the model.

Answer:

From the above LIME 'explanations', we observe the following:

1. First and foremost, as an aside ***(but a critical one),*** we note that ***our pre-processing steps seem to have done too much!*** Many of the *tidy_tweets* have simply become ***too tidy to interpret.*** The fully pre-processed tweets have been so pared down that in many cases, presented with a given *tidy_tweet,* I would have classified it in the same way as the applied model did. In other words, in interpreting the above performance metrics, we need to compare them not to an ***arbitrary, absolute value of 100% for precision, recall, f1-score, BUT rather to human-level performance!*** After all, in this particular language classification task, we ***cannot expect the AI/ML to outperform us!***
2. In looking at the previously culled sample of 21 false positive predictions by the word2vec-based model as well as 21 false negative predictions, we see from the above that the BiLSTM corrected many of the former model's errors. Again, this improved level of performance is in line with our expectations, given the ***incorporation by BiLSTM of critical forward/reverse word sequence/context information,*** in contrast to the other models tested.

Answers to Rubric Questions:

1. How does the Naive Bayes Classifier work? What is posterior probability?

- Naive Bayes Classifiers are a type of classifier based on *Baye's theorem*, which states that the **posterior probability** (that is, *'revised' probability of an event **w** given an 'a priori' event **x**) is given by:*

$$P(\omega \mid x) = \frac{P(x \mid \omega) \cdot P(\omega)}{P(x)}$$

- Classification is determined by applying the following *simple rule to the calculated posterior probabilities:*
    - ***If:***

$$P(+ \mid x) \geq P(- \mid x)$$

    - ***classify as +; else classify as — .***
- Naive Bayes performs well in e.g., text classification and disease prediction. It is a *simple, efficient, linear classifier that* ***assumes data features are independent***, i.e., that joint feature probabilities can be calculated by multiplying individual feature probabilities. It also assumes that *classes are* ***linearly separable***. Both of these assumptions decrease computational/time complexity, and while they are often not stringently met in practice, surprisingly, naive Bayes can still yield good performance.
- ***Multivariate Bernoulli Naive Bayes*** (used in this notebook) applies to binary data, and assigns a value of 1 or 0 to every token in an *m*-dimensional 'document' (tweet) feature vector, where *m* is the vocabulary size and 1 or 0 corresponds to whether or not the word represented by a token occurs or does not occur in the 'document' (tweet).

2. What is the difference between stemming and lemmatization in NLP?

*(Please see previous answer above under **Task section: §4.7 "Answer Addendum: Observations on model performance, and considerations for improving predictions", sub-point 1: "Text pre-processing"**.)*

3. What is Word2Vec and how does it work?

*(Please see previous detailed answers above under **Task sections §3.8, §3.9, and §4.7, sub-point 4: "Model Limitations"**.)*

4. When to use GRU over LSTM?

- LSTMs (long-short term memory units) are specialized RNN units which employ 3 gates (Input, Forget, and Output) at each sequence step to enable capturing long-range dependencies in sequences e.g., text (example: remembering that a plural subject at the beginnning of a sentence ought to be matched with a plural verb form later on), while avoiding the vanishing gradient problem that can arise from lengthy sequences.
- Similarly, GRUs (gated-recurrent units) are specialized RNN units which also attempt to capture long-range dependencies while avoiding the vanishing gradient problem, but use only two gates (Update and Reset).
- LSTMs preceded GRUs and are a more generalized implementation with a proven track record, while GRUs were developed much later as a simplification of LSTMs. Thus, GRUs scale better to larger neural networks. ***Generally speaking, the more compute intensive LSTM is chosen when dealing with large sequences and optimal accuracy is desired, while GRUs are preferred for speed and lighter hardware requirements (e.g. RAM).***

Acknowledgement & Reference

- Data is adapted from Twitter sentiment analysis
- Twitter sentiment analysis
- Introduction to Word Embedding and Word2Vec
- When to use GRU over LSTM?
- Use a trained Word2Vec, Doc2Vec or FastTest embedding by `gensim` in buiding an embedding layers in Tensorflow, here's how-to
- ref 1: gensim word2vec print log loss

Colab paid products - Cancel contracts here