Toxic Comment Classifier Report

-Lavanya Samanta

AIM:

To build a model that's capable of classifying tweets as toxic or clean.

INTRODUCTION:

Trained a TensorFlow model using word-embeddings and keras, libraries like the natural language toolkit, pandas and regular expression for the pre-processing of data, seaborn and matplotlib for visualization and data analysis. Attained an accuracy score of 0.953

METHODS:

1. The original dataset:

- Count: Number of people who coded each tweet
- hate_speech: Number of people who judged tweet to be hate speech
- offensive_language: Number of people who judged tweet to be offensive language
- neither: Number of people who judged tweet as neither hate nor offensive
- class: 0 = hate_speech, 1 = offensive_language, 2 = neither

Removed all rows with hate speech label due to vast imbalance in the dataset.

```
df.head(5)
```

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

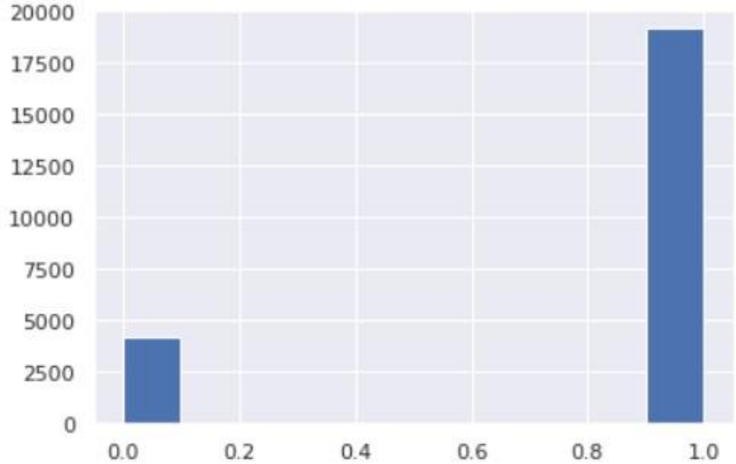
2. Now the dataset to be trained consists of 2 labels in class. Relabelled offensive language label as 1 and clean label as 0.
   There is still an imbalance in the dataset with 19190 toxic tweets and 4163 clean tweets.

```
df #clean label=0 ; toxic label = 1
```

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	0	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...
...	...	...	...	...	...	...	...
24778	25291	3	0	2	1	1	you's a muthaf***in lie “@LifeAsKing: @2...
24779	25292	3	0	1	2	0	you've gone and broke the wrong heart baby, an...
24780	25294	3	0	3	0	1	young buck wanna eat!!.. dat nigguh like I ain...
24781	25295	6	0	6	0	1	youu got wild bitches tellin you lies
24782	25296	3	0	0	3	0	~~Ruffled | Ntac Eileen Dahlia - Beautiful col...

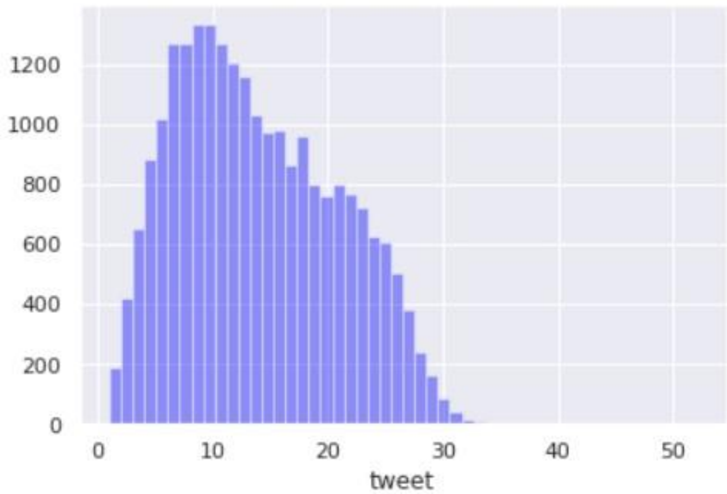
23353 rows × 7 columns

```
Label 0: Clean tweets
Label 1: Toxic tweets
```



3. Number of words in each tweet ranges within 1-35 words.

```
Number of words in each tweet
<matplotlib.axes._subplots.AxesSubplot at 0x7f1e381a8128>
```



4. Pre-processing the data:
   a. Replace all long whitespaces, URLs and mentions with a single whitespace using regular expression.
   b. All the numbers and punctuation marks in the comment text section were replaced with a whitespace and then converted to lower case. The sentences were split about the whitespaces to remove the stop words. The function returns a processed string.

```
def preprocess(text):
    corpusTrain = []
    space_pattern = '\s+'
    giant_url_regex = ('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|''[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
    mention_regex = '@[\w\-]+'
    parsed_text = re.sub(space_pattern, ' ', text)  #replace lots of whitespace with a single whitespace
    parsed_text = re.sub(giant_url_regex, '', parsed_text) #replace url with ''
    parsed_text = re.sub(mention_regex, '', parsed_text)   #replace @mention with ''
    review = re.sub("[^a-zA-Z]", " ", parsed_text)
    review = review.lower()
    review = review.split(" ")
    review = [word for word in review if word not in set(stopwords)]
    review = (" ").join(review)
    review = re.sub(space_pattern, ' ', review)
    return review
```

```
df
```

	Unnamed: 0	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	0	woman complain cleaning house amp man always ...
1	1	3	0	3	0	1	boy dats cold tyga dwn bad cuffin dat hoe st ...
2	2	3	0	3	0	1	dawg ever fuck bitch start cry confused shit
3	3	3	0	2	1	1	look like tranny
4	4	6	0	6	0	1	shit hear might true might faker bitch told ya

5. The vocabulary is "trained" on a corpus and all word pieces are stored in a vocabulary file, encoder.

```
encoder = tfds.features.text.SubwordTextEncoder.build_from_corpus(
    df.tweet, target_vocab_size=19241)
```

```
encoder.subwords[:20]
```

```
['bitch_',
 'bitches_',
 'like_',
 'hoes_',
 'pussy_',
 'hoe_',
 'ass_',
 'get_',
 'fuck_',
 'got_',
 'u_',
 'bitch',
 'shit_',
 'nigga_',
 'amp_',
 'trash_',
 'know_',
 'niggas_',
 'one_',
 'love_']
```

6. Tokenize the tweets with most frequent maximum number of words as 5000 and pad the sequences with maximum sequence length as 50.

```
# The maximum number of words to be used. (most frequent)
MAX_NB_WORDS = 5000
# Max number of words in each complaint.
MAX_SEQUENCE_LENGTH = 50
tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~' '', lower=True, oov_token='OOV')
tokenizer.fit_on_texts(df['tweet'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 19241 unique tokens.
```

```
X = tokenizer.texts_to_sequences(df['tweet'].values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)
```

```
Shape of data tensor: (23353, 50)
```

```
Y = pd.get_dummies(df['class']).values
print('Shape of label tensor:', Y.shape)
```

```
Shape of label tensor: (23353, 2)
```

7. The labels are one hot encoded hence we use categorical cross entropy as loss function.

8. Model:

- We will use the Keras Sequential API to define our model.
- Next the Embedding layer takes the integer-encoded vocabulary and looks up the embedding vector for each word-index. These vectors are learned as the model trains.
- Next, a GlobalAveragePooling1D layer returns a fixed-length output vector for each example by averaging over the sequence dimension. This allows the model to handle input of variable length, in the simplest way possible.
- This fixed-length output vector is piped through a fully-connected (Dense) layer with 6 hidden units.
- The last layer is densely connected with 2 output nodes. Using the SoftMax activation function, the values are either a 0 or 1 for respective label.

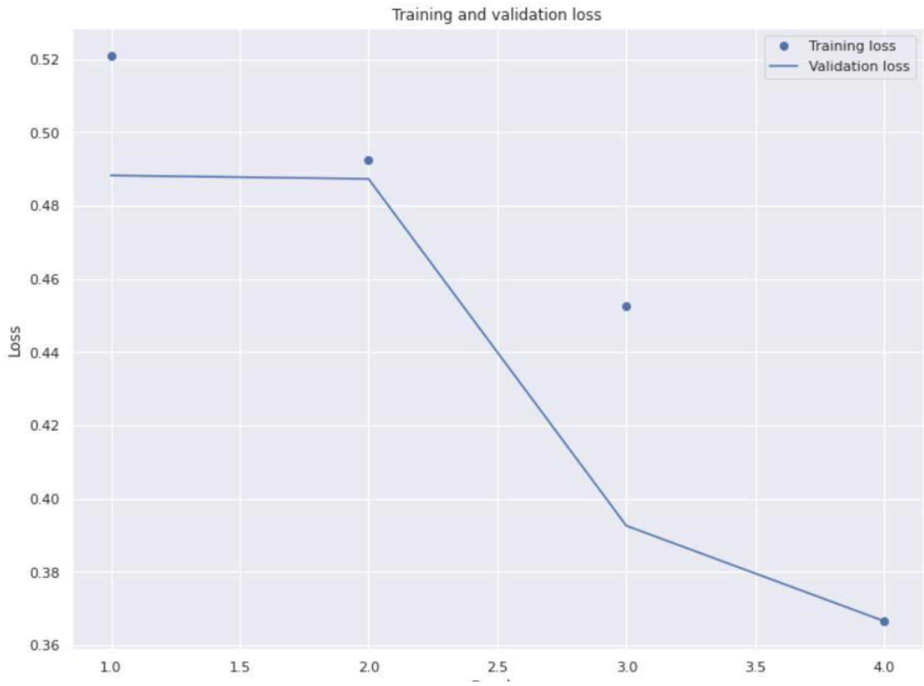
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 50, 50)            1102400
_________________________________________________________________
global_average_pooling1d (Gl (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 6)                 306
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 14
=================================================================
Total params: 1,102,720
Trainable params: 1,102,720
Non-trainable params: 0
```

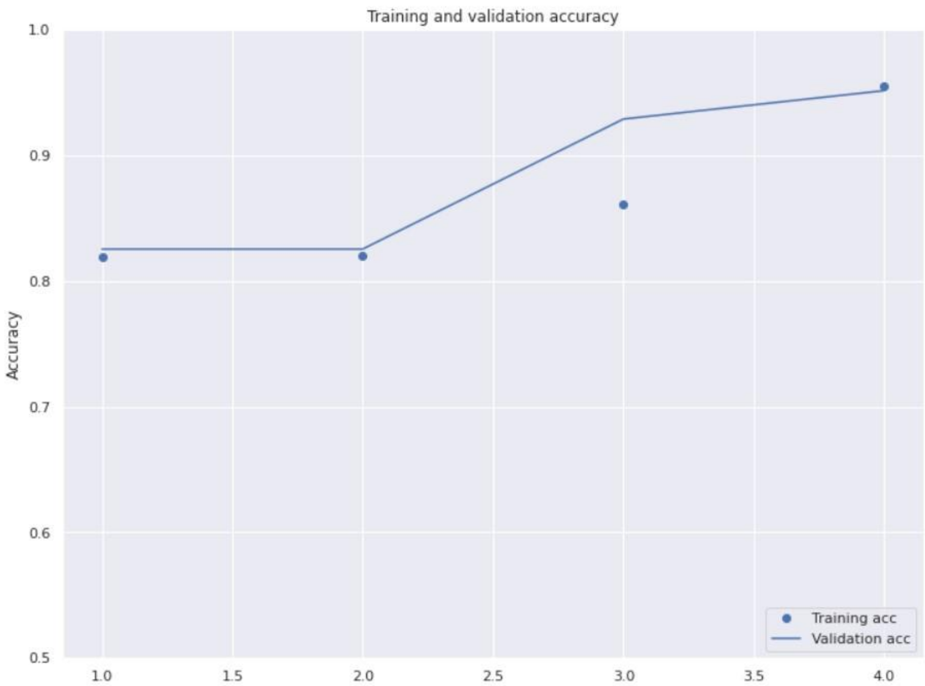
9. Training:

```
model.compile(optimizer='adam', loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=4, validation_data=(X_val,Y_val), validation_steps=20)
```

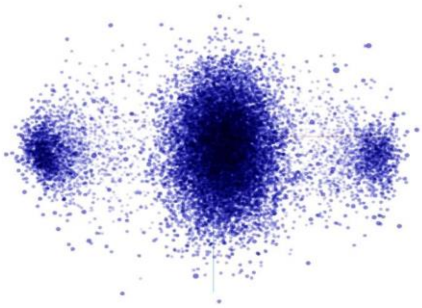
```
Epoch 1/4
526/526 [==============================] - 7s 13ms/step - loss: 0.5209 - accuracy: 0.8195 - val_loss: 0.4883 - val_accuracy: 0.8254
Epoch 2/4
526/526 [==============================] - 6s 12ms/step - loss: 0.4925 - accuracy: 0.8207 - val_loss: 0.4873 - val_accuracy: 0.8254
Epoch 3/4
526/526 [==============================] - 7s 14ms/step - loss: 0.4525 - accuracy: 0.8614 - val_loss: 0.3927 - val_accuracy: 0.9289
Epoch 4/4
526/526 [==============================] - 7s 12ms/step - loss: 0.3667 - accuracy: 0.9547 - val_loss: 0.3666 - val_accuracy: 0.9515
```

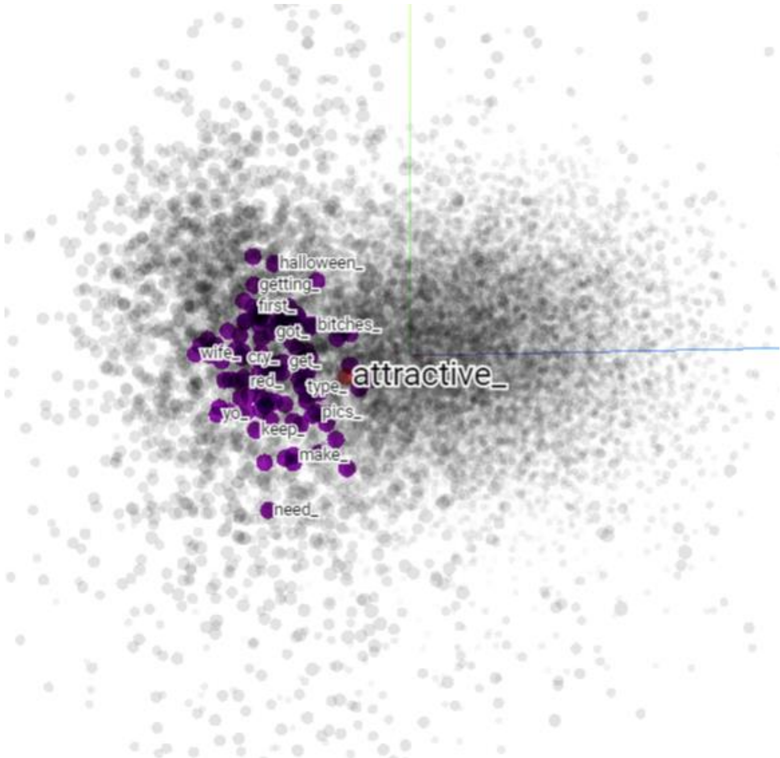
10. Graphs:





11. Store the weights from the model layer. Get the metadata and tensors tsv file to visualize the embedding.





CONCLUSION:

```
accr = model.evaluate(X_test,Y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
73/73 [==============================] - 0s 1ms/step - loss: 0.3663 - accuracy: 0.9533
Test set
  Loss: 0.366
  Accuracy: 0.953
```

The model attained an accuracy score of 0.953 without overfitting.