

The Myers Briggs personalities of harassment victims are predicted.

using BiDirectional LStm for classification

Myers Briggs Personality Detection

Personality Detection

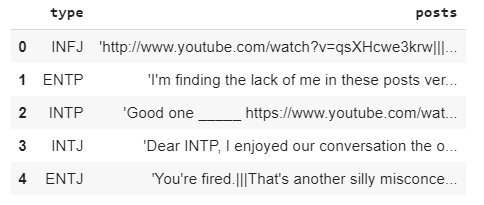
USING BIDIRECTIONAL LSTM FOR CLASSIFICATION

[Type the sidebar title]

[Type the sidebar content. A sidebar is a standalone supplement to the main document. It is often aligned on the left or right of the page, or located at the top or bottom. Use the Drawing Tools tab to change the formatting of the sidebar text box.]

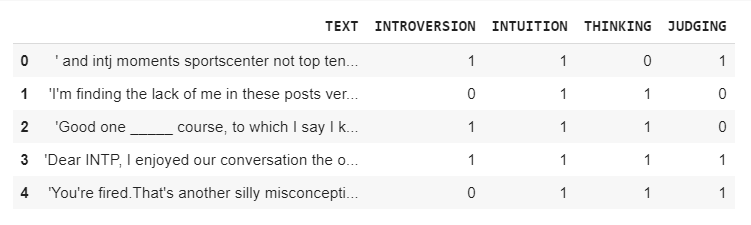
**Reading the CSV:**

The csv file was read and the data were transferred into a pandas dataframe. There were two columns in the data ‘Posts’ and ‘Type’ respectively.



**One Hot Encoding:**

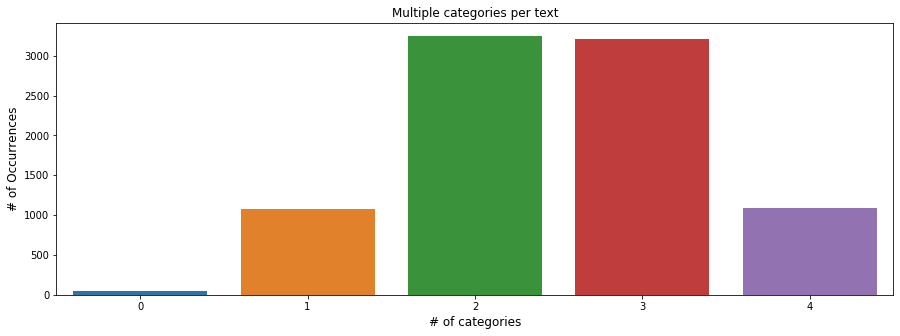
The One Hot Encoding is a technique in which the traits present in a particular post are set as 1 and the traits absent are marked as 0. The columns are added into the dataframe and the previous columns of ‘Posts’ and ‘Type’ were dropped. It is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.

****

**Separating the Data and Labels:**

The Labels for each personality trait were separated into a Labels Dataframe while the Text was kept in a separate array.

The when divided according to Labels are as:



**Creating the Model:**

From the tensorflow.keras.preprocessing.text module, we imported Tokenizer

## Tokenizer:

This class allows to turn each text into a sequence of integers, each integer being the index of a token in a dictionary.

Arguments:

num\_words: the maximum number of words to keep, based  
    on word frequency. Only the most common `num\_words-1` words will  
    be kept.

oov\_token: if given, it will be added to word\_index and used to  
    replace out-of-vocabulary words during text\_to\_sequence calls

Explanation:

We have set the num\_words to 1000, it means that it will retain only the 1000 most common in our data will be kept and the rest would be considered out of dictionary.

OOV Token or <oov\_tok> is the token used for words that are not included in our dictionary instead of discarding the very word from sequence.

Eg. Let’s suppose xoxo is not included in dictionary. That’d be:

Without the oov token,

Let’s meet sometime, xoxo

Let’s meet sometime,

0 1 2 3

With the oov token,

Let’s meet sometime, xoxo

Let’s meet sometime, <oov\_tok>

0 1 2 3 <oov\_tok>

In our code:

tokenizer = Tokenizer(num\_words=vocab\_size, oov\_token=oov\_tok)

From the tensorflow.keras.preprocessing.sequence module, we imported pad\_sequences

## Pad Sequences:

Sequences that are shorter than num\_timesteps are padded with value at the end.

Sequences longer than num\_timesteps are truncated so that they fit the desired length. The position where padding or truncation happens is determined by the arguments padding and truncating, respectively.

### Arguments:

sequences: List of lists, where each element is a sequence.  
maxlen: Int, maximum length of all sequences.

padding: String, 'pre' or 'post':  
    pad either before or after each sequence.  
truncating: String, 'pre' or 'post':  
    remove values from sequences larger than  
    `maxlen`, either at the beginning or at the end of the sequences.

Sequence is the sequence on which padding is to be applied.

Maxlen is the most important argument, as it decides the length of sequence after padding. If the length of sequences is less than 1000, it will add 0s at the end to reach the desired length, else it will truncate the sequence up to that length.

If the padding is set to ‘pre’ it will pad before the start of sequence and if it is set to ‘post’ it will pad after the sequence has ended.

If the truncating is set to ‘pre’ it will cut the starting sequences that lies outside the maxlen range and vice versa.

vocab\_size = 1000---------------------Vocab size to be used in Tokenizer

embedding\_dim = 16--------------------Embedding Dimensions for Embedding Layer.

max\_length = 120----------------------Max length for Padding

trunc\_type='post'---------------------Truncation Type for Padding

padding\_type='post'-------------------Padding Type for Padding

oov\_tok = "<OOV>"---------------------OOV Token for Padding

training\_size = int(len(X)\*0.8)-------Training size for Training Data

word\_index = tokenizer.word\_index-----The Dictionary containing word, index map

## Model:

Our model consists of 5 layers:

1. Embedding
2. Bi-directional LSTM
3. Dense
4. Dropout
5. Dense (2)

### Embedding:

### Arguments:

**input\_dim**: int > 0. Size of the vocabulary, i.e. maximum integer index + 1.

**output\_dim**: int >= 0. Dimension of the dense embedding.

**input\_length**: Length of input sequences, when it is constant. This argument is required if you are going to connect Flatten then Dense layers upstream.

So, what happens in an Embedding Layer?

It takes as input the size of dictionary and max\_length of sequence (the padded one). It will map each number in sequence to a vector that would be defining its meaning. (Just like in real life dictionaries we have word to meaning mapping. Here we have number to vector mapping.) The size of a single Embedding would be of size of embedding dimension that we define.

Input\_dim / vocab\_size=10,

input\_length = 5,

output\_dim = 4

Example:

Ali likes apple

After Padding:

2 3 4 0 0

In Embedding Layer:

2 3 4 0 0

[0.2, 0.4, 0.21, 0.31] [0.98, 0.21, 0.32, 0.54] [0.23, 0.43, 0.56, 0.34] [0.0, 0.0, 0.0, 0.0] [0.0, 0.0 ,0.0 ,0.0]

Output Shape=

1x4 + 1x4 + 1x4 + 1x4 + 1x4

= 5 x 4

In our Example:

Input\_dim / vocab\_size=1000,

input\_length = 120

output\_dim = 16

2 3 4 5 …. 120

[0.2, 0.4, 0.1, 0.1, 0.9, 0.2, 0.3, 0.1, 0.9, 0.2, 0.3, 0.1, 0.9, 0.2, 0.3, 0.1] [0.9, 0.2, 0.3, 0.1, 0.9, 0.2, 0.3, 0.1,0.8, 0.2, 0.3, 0.5, 0.9, 0.2, 0.3, 0.1] [0.2, 0.4, 0.1, 0.1, 0.9, 0.2, 0.3, 0.1, 0.9, 0.2, 0.3, 0.1, 0.9, 0.2, 0.3, 0.1] [0.3, 0.4, 0.5, 0.4, 0.2, 0.4, 0.1, 0.3, 0.9, 0.2, 0.2, 0.4, 0.2, 0.7, 0.1, 0.9] … [0.9, 0.2, 0.3, 0.1, 0.9, 0.2, 0.3, 0.1, 0.9, 0.2, 0.3, 0.1, 0.9, 0.2, 0.3, 0.1, 0.9, 0.2, 0.3, 0.1 ]

Output Shape=

1x16 + 1x16 + 1x16 + 1x16 +

... + 1x16

= **120 x 16**

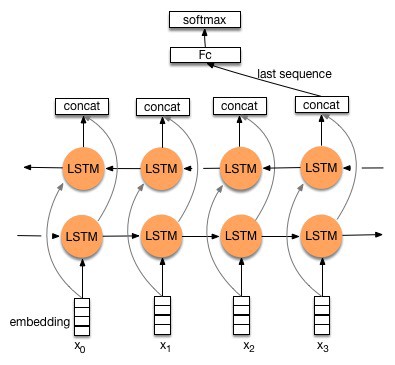
### Bi-Directional lstm:

LSTM works by traversing the sentences and extracting the whole meaning out of it. And the Bi Directional LSTM would help getting the context of the word based on complete sentence.   
Just like this apple example.

I found Apple very tasty and healthy.

I found Apple MacBook quite convenient for office work.

Here, we have two examples of Apple and both are in different context and with Bi LSTMs we can treat both of them as different entities.



### Arguments:

**units**: Positive integer, dimensionality of the output space.

### Dense:

### Arguments:

**units**: Positive integer, dimensionality of the output space.

**activation**: Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x)

A dense layer is just a regular layer of neurons in a neural network. Each neuron receives input from all the neurons in the previous layer, thus densely connected. The layer has a weight matrix **W,**a bias vector **b,**and the activations of previous layer **a.**

**Activation Function:**

In every Dense Layer of a Neural Network, there is an output. If that’s the hidden layer the output of layer is the input to next layer and so on.

The output is affected by an activation function. You can read more at:

<https://www.deeplearning-academy.com/p/ai-wiki-activation-functions>

And the use of activation functions is defined in this lecture:

<https://www.coursera.org/lecture/neural-networks-deep-learning/activation-functions-4dDC1>

### DROPOUT:

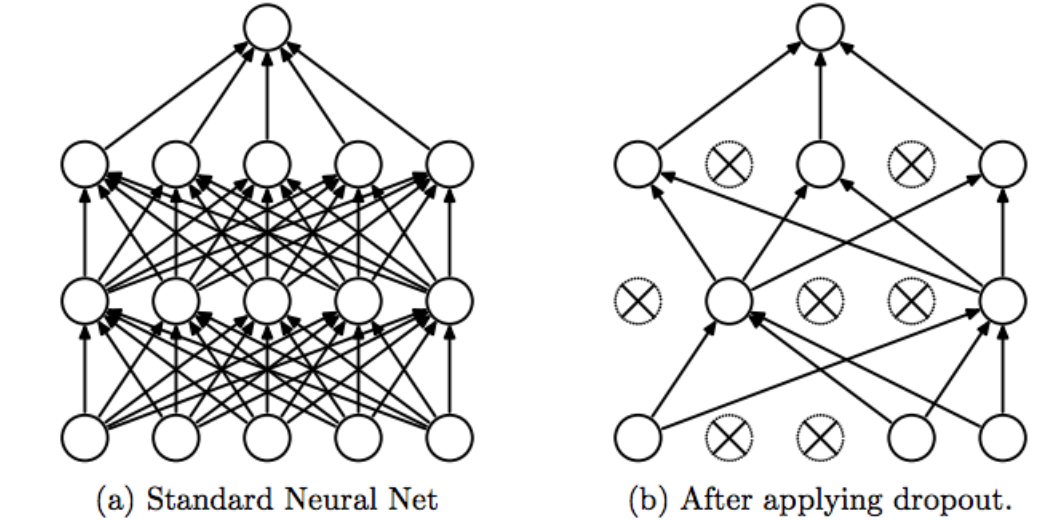
### Arguments:

**rate**: Float between 0 and 1. Fraction of the input units to drop.

In Dropout Layer we drop some random layers from Input to avoid overfitting. The rate as a parameter tells how much units we want to drop.

OVERFITTING:

When we get pretty high accuracy in training but the validation or test accuracy decrease as training accuracy increase, this is called overfitting. It means that the model is fitting perfectly on provided data so much that it is not accepting variations. For example: if model fits very well on heels and fit on heels as shoes it may fail to accept Boots are shoes.



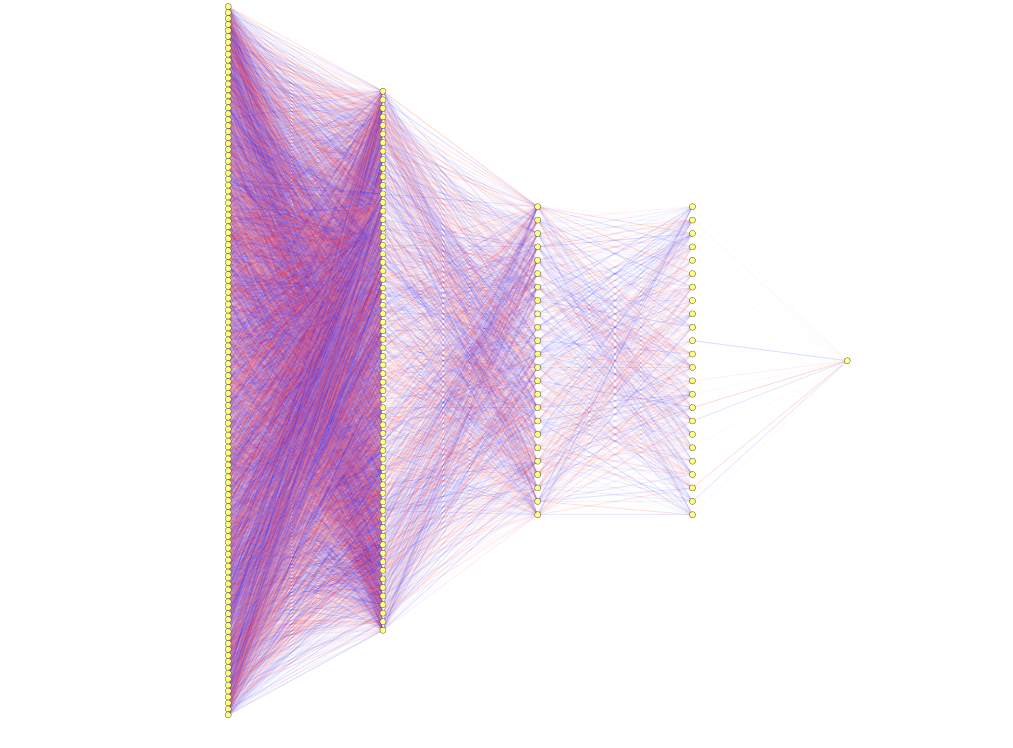
## Epochs:

We divide our Total Training and Test Data into certain batches. Since deep learning often separates training data into smaller batches when training, it is important to know when all the training examples have been processed a single time. This is called an epoch.

One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.

Increasing the number of epochs should increase accuracy.

## Here’s what our model looks like:



## Model.fit:

This is the phase in which we train our data.

### Arguments:

verbose=1 will show you an animated progress bar like this:

[progres_bar](https://i.stack.imgur.com/s43II.png)

<https://www.tensorflow.org/api_docs/python/tf/keras/Model>

After the model has been trained. We tested the accuracy of model using model.evaluate() . You can read more about it from above link.

## Predictions:

After that we made certain predictions with our data from training set. We also did predictions from some sentences provided.

## Extracting the person:

We extracted the victims and harassers from the file provided. And added them to respective arrays.

## Extracting the chats:

Then we extracted the chats and extracted the lines that were from victims. We cleaned the data and removed slangs. The text was then cleaned further and made readable.

We then had chat as a String for each document/ chat separately. The chat was stored in lines of appropriate lengths and predicted.