Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Winter Semester 2022 – 2023

CSE3021- Social and Information Network

Deep learning-based personality recognition from text posts of online social networks

SUNNY KUMAR

20BCE0262

SLOT:C1 Slot

FACULTY:-Dr. RAJKUMAR R

Objective

Social networks such as Facebook, Twitter, and Weibo have become essential components of everyday life and hold rich sources that reflect an individual's personality.

Many approaches have been proposed to automatically infer users' personality fromthe content they generate in social networks. However, the performance of these approachesdepends heavily on the data representation which often is based on hard-coded prior knowledge. Recently, deep learning approaches have obtained very high performance.

Scope

Our daily lives are filled with concerns that relate to the assessment and prediction ofpersonality. All social interaction requires that we evaluate and try to predict the behaviour of other persons with whom we must deal.

If an individual's personality could be predicted with a little more reliability, there is scope for integrating automated personality detection in almost all agents dealing withhuman-machine interaction such as voice assistants, robots, cars, etc. Research in this field is moving from detecting personality solely from textual data to visual and multimodal data

Abstract

Many approaches have been proposed to automatically infer users' personality fromtheir social networks activities. However, the performance of these approaches depends heavily on the data representation. In this work, we apply deep learning methods to automatically learn suitable data representation for the personality recognition task.

Literature Survey

S.No	Title	Source	Year of publication	Findings of the research work	Limitations	Future Work
1	Affective Computing and Sentiment Analysis	https://ieeexplore.ieee.org/document/7435182	17 March 2016	listing common tasks of affective computing and sentiment analysis and present a general categorization for them, they found that analysis falls into three main categories: knowledge based techniques, statistical methods, and hybrid approaches.Knowledge-based techniques are popular because of their accessibility and economy. Statistical methods, such as support vector machines and deep learning,	The major weakness of knowledge based approaches is poor recognition of affect when linguistic rules are involved.statistical methods are generally semantically weak—that is, lexical or co-occurrence elements in a statistical model have little predictive value individually	Researching more on Next-generation sentiment-mining systems need broader and deeper common and common sense knowledge bases, together with more brain inspired and psychologically motivated reasoning methods, to better understand natural language opinions and, hence, more efficiently bridge the gap between (unstructured) multimodal information and (structured) machine-processable data.

				have been popular for affect classification of texts, Hybrid approaches aim to better grasp the conceptual rules that govern sentiment and the clues that can convey these concepts from realization to verbalization in the human mind		
2.	Deep Learning-Based Document Modeling for Personalit y Detection from Text	https://sentic.net/deep-learning-based-personality-detection.pdf	2017	a method to extract personality traits from stream of-consciousness essays using a convolutional neural network (CNN). method outperformed the state of the art for all five traits, although with different configurations for	Using n-grams showed no improvement over the majority baseline: the classifier rejected all n-grams. SVM to the document vector d built with the CNN	incorporate more features and preprocessing. We plan to apply the Long Short Term Memory (LSTM) recurrent network to build both the sentence vector from a sequence of word vectors and the document vector from a sequence of sentence vectors. In addition, we plan to apply our document modeling technique to other

				different traits. Using n-grams showed no improvement over the majority baseline: the classifier rejected all n-grams. Applying filtering and adding the document level (Mairesse) features proved to be beneficial. In fact, the CNN alone without the document-level features underperformed the Mairesse baseline.	did not improve the results	emotion related tasks, such as sentiment analysis or mood classification
3.	Semantic Analysis to Compute Personality Traits from Social Media Posts	https://giulioc.github.io/files/Semantic_analysis_to_Compute_Personality_Traits_from_Social_	2017	Semantic Analysis to Compute Personality Traits from Social and study scientific aspects of the analysis, namely, whether it is possible to accurately predict someone's	The approach has only been tested on English language and might not show the same results for other languages	plan to extend it to other languages exploiting the similarity of word meanings in the vector space. Finally, they aim to extend the Twitter sample by acquiring more panelists running our questionnaire.

		Media_Posts.pdf		personality by only using the language features presented in a social network context.SVM performs better than other algorithms. For each personality trait exists a SVM configuration with a minimum MSE lower than that of other learning models		
4.	Personality Recognition on Social Media With Label Distribution Learning	https://ieeexplore.ieee.org/abstract/document/7956172	23 June 2017	present a novel methodology of personality recognition based on a new machine learning paradigm named label distribution learning (LDL), which assigns a label distribution rather than a single label or a relevant label set to each instance.the	disregard the performance differences among different LDL approaches, and only focus on the MAEs of the approaches with same LDL algorithm, Dataset was not large enough to get	plan to explore optimal feature space by applying deep learning techniques on larger data sets so as to further improve the prediction accuracy of the LDL approaches in personality recognition.

				running efficiency of PR approach is also significant since high-efficiency ones could be more adaptive for various application scenarios.t LDL algorithms are able to predict all the Big Five traits of a given user at once, while the baselines need to build five independent prediction models, one for each trait	better accuracy.	
5.	Cross-platform and cross-interaction study of user personality based on images on Twitter	https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6040697/	2018 Jul 11	The aim of this paper is to quantify image sharing preferences and to build models that automatically predict users' personality in a cross-/modal and cross-platform setting.Overall,	This work is not optimized for different situations dependent on platform and purpose of use of the social network by the client for example	Implementation and analysis of models on domains like A lot of systems can benefit from personality detection. For example, dating websites can trying to match personalities of individuals before they meet each other . Human Resources department could predict job

	and Flickr			our analysis shows that conscientiousness and openness to experience are the most predictable personality traits from images posted online.	facebook is supposed to be a casual platform where as linkedIn is supposed to be a professional one.	satisfaction before hiring a potential employee. Recommendation systems and commercial companies can improve their accuracy by recommending photos, movies or music, that have higher chance to make positive impressions on their users. Knowledge of a user's personality also enables software developers to customise user interfaces

Overall Description

Language is the most common and reliable way for people to translate their internal thoughts and emotions into a form that others can understand. Words and language, then, are the very stuff of psychology and communication. Texts tend to reflect various aspects of the author's personality, and if we could model the user's text posts better, the performance of PR approaches would improve a lot. Motivated by this intuition, we propose three deep learning sequential models - single layer LSTM model, single layer 1D Convolution model and a double layer 1D convolution model. We can then analyse and compare the results of the three proposed models to conclude which model gives the most accurate results for the problem statement in hand. Then we concatenate them with pre-extracted global statistical features toconstruct the input feature space for the traditional regression algorithm to carry out final prediction of each user's real-valued Big Five personality scores.

We have used various data cleaning techniques to improve the accuracy of the model along with analysis of the data with various graphs to give us a better understanding of the data thatwe are dealing with which helps us develop a better model for classifying the posts into 16 different personality types. Our study even gives a general idea about whether just increasingthe complexity of the model used for training yields better accuracy or not. This project incorporates multiple callback functions to avoid common machine learning issues such as overfitting of data and saving the most recent model instead of the most efficient model.

Implementation

1. Importing the dataset

The dataset used in this project is the (MBTI) Myers-Briggs Personality Type Dataset from kaggle.com , this dataset contains over 8674 records which are divided into twocolumns 'type' which is the personality type of the person and 'posts' which is a text type data which consists of all the posts of the person.

```
import pandas as pv
df = pv.read_csv('mbti_1.csv')
df.head()
```

	type	posts
0	INFJ	'http://www.youtube.com/watch?v=qsXHcwe3krw|||...
1	ENTP	'I'm finding the lack of me in these posts ver...
2	INTP	'Good one _____ https://www.youtube.com/wat...
3	INTJ	'Dear INTP, I enjoyed our conversation the o...
4	ENTJ	'You're fired.|||That's another silly misconce...

```
df=df.dropna()
df.reset_index(inplace=True)
```

2. Cleaning the data

As we can see above the data consist of many redundancies like email IDs, HTML tags, URLs, stopwords which play no role in solving the problem in hand , another issue with the dataset is that a few abbreviations with the same meaning as the non abbreviated version are considered as different words which makes the model inefficient . Therefore to solve these problems cleaning the data in necessary beforetrain the model. In this project we have used text-hammer library from python by calling the necessary functions and cleaning the data.

\+ Code   + Text

```
import text_hammer as th
import regex as re
def get_clean(x):
    x = str(x).lower()
    x = re.sub(r"\|\|\|", " ", x)
    x = th.cont_exp(x)
    x = th.remove_emails(x)
    x = th.remove_urls(x)
    x = th.remove_html_tags(x)
    x = th.remove_stopwords(x)
    x = th.remove_rt(x)
    x = th.remove_accented_chars(x)
    x = th.remove_special_chars(x)
    # x = th.spelling_correction(x)
    x = th.make_base(x)

    return x


df['posts'] = df['posts'].progress_apply(
    lambda x: x.replace(x, get_clean(x)))

df.to_csv('Others\mbti_1.csv', index=False)
```

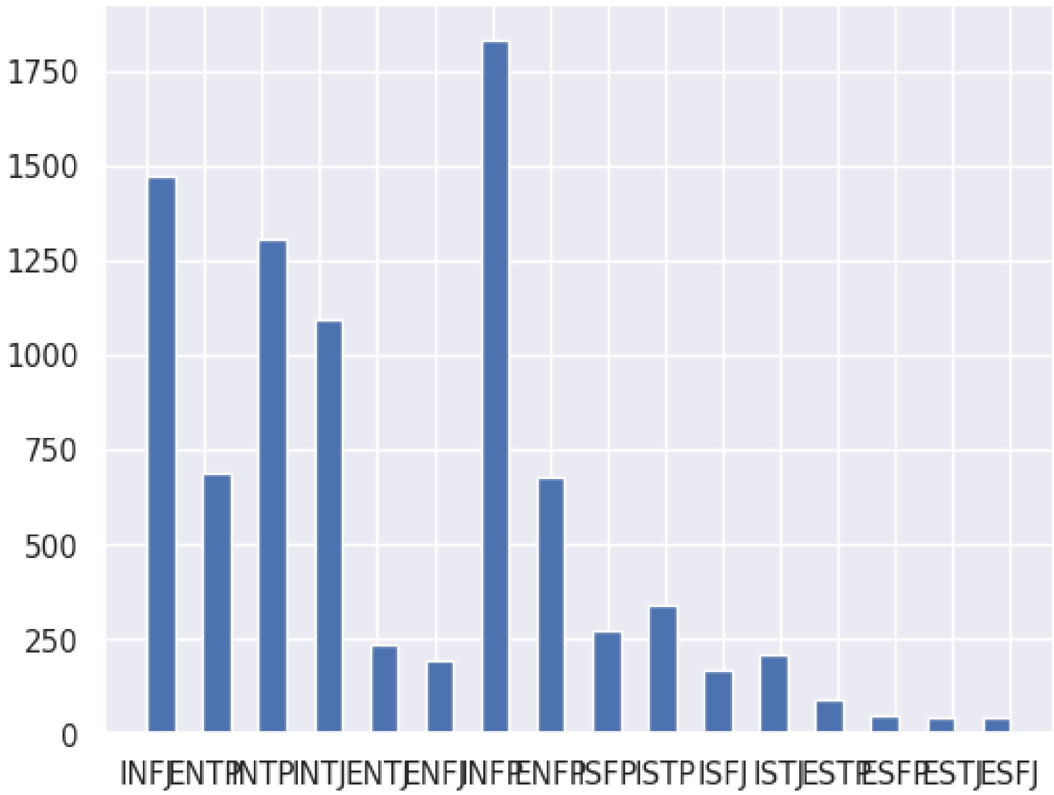
```
100% 8675/8675 [24:49<00:00, 6.16it/s]
```

3. Data Visualisation

Data Visualisation is an important part of any Machine Learning based or deep learning based problem, it gives an overview and insights of our data which we cannot infer just by looking at our data , In this project we have tried to visualise the data using various plots and graphs :

1. Frequency bar plot of various personality types

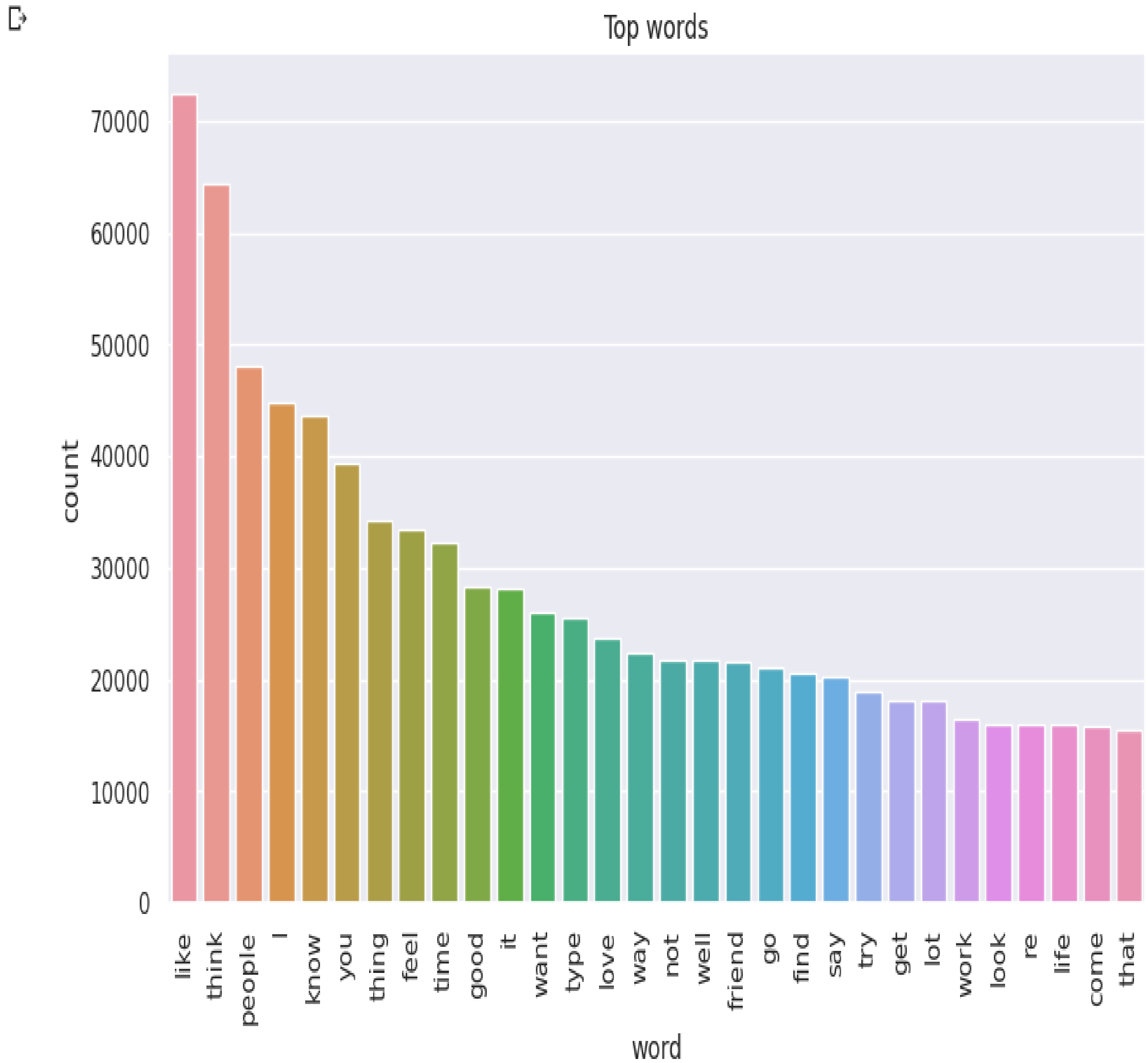
```
[ ]  import seaborn as sns
     import matplotlib.pyplot as plt
     # sns.set()
     # df['type'] = pv.Categorical(df['type'])
     # sns.countplot(df.iloc[:,1].values)
     # sns.countplot(df['type'])
     # plt.show()
     # df.iloc[:,1].hist(bins=32)
     plt.hist(df.iloc[:, 1].values, bins=31)
     plt.show()
```



2. Frequency bar plot of top words used in posts

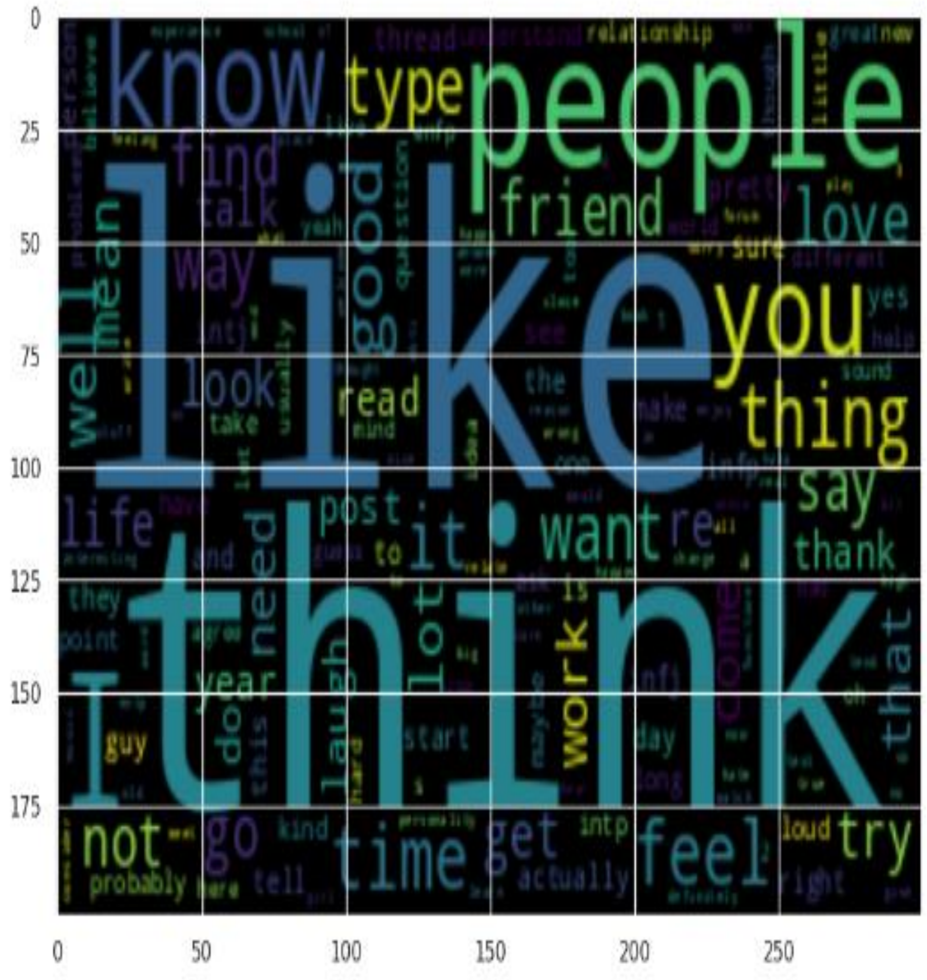
```
[ ]  import matplotlib.pyplot as plt
     import seaborn as sns
     import nltk
     import pandas as pd

     nltk.download('punkt')

     words_list= []

     for post in df['posts']:
         words_list.extend(nltk.word_tokenize(post))

     freq_dist = nltk. FreqDist (words_list)
     freq_dist.most_common(20)
     temp=pd.DataFrame(freq_dist.most_common (30), columns=['word', 'count'])

     fig, ax = plt.subplots(figsize=(10, 6))
     sns.barplot(x='word', y='count',data=temp, ax=ax)

     plt.title("Top words")

     plt.xticks(rotation = 'vertical')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]),
 [Text(0, 0, 'like'),
  Text(1, 0, 'think'),
  Text(2, 0, 'people'),
  Text(3, 0, 'I'),
  Text(4, 0, 'know'),
  Text(5, 0, 'you'),
  Text(6, 0, 'thing'),
  Text(7, 0, 'feel'),
  Text(8, 0, 'time'),
  Text(9, 0, 'good'),
  Text(10, 0, 'it'),
  Text(11, 0, 'want'),
  Text(12, 0, 'type'),
  Text(13, 0, 'love'),
  Text(14, 0, 'way'),
  Text(15, 0, 'not'),
  Text(16, 0, 'well'),
  Text(17, 0, 'friend'),
  Text(18, 0, 'go'),
  Text(19, 0, 'find'),
  Text(20, 0, 'say'),
  Text(21, 0, 'try'),
  Text(22, 0, 'get'),
  Text(23, 0, 'lot'),
  Text(24, 0, 'work'),
  Text(25, 0, 'look'),
  Text(26, 0, 're'),
  Text(27, 0, 'life'),
  Text(28, 0, 'come'),
  Text(29, 0, 'that')])
```


Top words
count
70000
60000
50000
40000
30000
20000
10000
0
like
think
people
I
know
you
thing
feel
time
good
it
want
type
love
way
not
well
friend
go
find
say
try
get
lot
work
look
re
life
come
that
word

3. WordCloud of the words used in the dataset

```python
import wordcloud
from wordcloud import WordCloud

# creation of wordcloud

wcloud_fig = WordCloud (stopwords=set (wordcloud.STOPWORDS),colormap='viridis', width=300, height=200).generate_from_frequencies (freq_dist)

# plotting the wordcloud

plt.figure(figsize=(10,7), frameon=True)
plt.imshow(wcloud_fig, interpolation = 'bilinear')
plt.show()
```



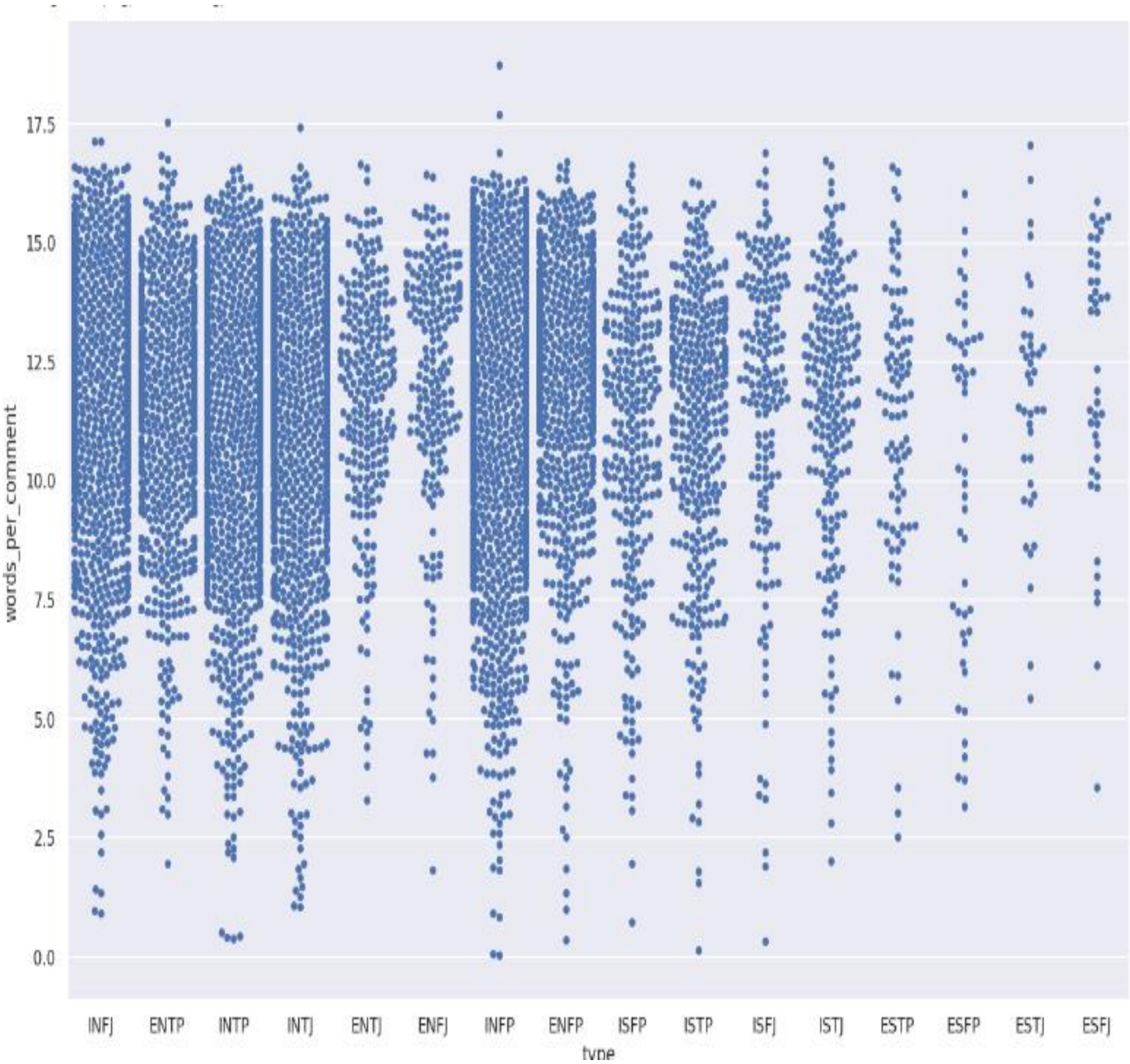
4. Swarmplot of number of words per comment of each type

```
[ ] df['words_per_comment']=df['posts'].apply(lambda x:len(x.split())/50)
    df.head()
```

	index	type	posts	words_per_comment
0	0	INFJ	enfp intj moment sportscenter play prank lifec...	5.94
1	1	ENTP	I find lack post alarm sex boring position oft...	11.06
2	2	INTP	good _ _ _ _ _ course know blessing curse abso...	8.58
3	3	INTJ	dear intp enjoy conversation day esoteric gabb...	10.34
4	4	ENTJ	you re fire silly misconception approach logic...	9.26

```
[ ] plt.figure(figsize=(15,10))
    sns.swarmplot(x="type",y= "words_per_comment", data=df)
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 68.4% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 45.0% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 64.1% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 58.4% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 6.8% of the points cannot be pl
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 73.1% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 45.5% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 10.0% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 14.8% of the points cannot be p
  warnings.warn(msg, UserWarning)
<Axes: xlabel='type', ylabel='words_per_comment'>/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 43.9% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 63.2% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 57.4% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 5.8% of the points cannot be pl
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 71.9% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 44.4% of the points cannot be p
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 8.5% of the points cannot be pl
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 13.9% of the points cannot be p
  warnings.warn(msg, UserWarning)
```



words_per_comment
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
INFJ
ENTP
INTP
INTJ
ENTJ
ENFJ
INFP
ENFP
ISFP
ISTP
ISFJ
ISTJ
ESTP
ESFP
ESTJ
ESFJ
type

4. Encoding the personality types

The personality types currently are in string type data in the dataset , but our modelcan only work with numerical type data so we will convert the array of string type data into a array of arrays with the length 16 as following:

```
[ ] import tensorflow as tf
    from tensorflow.keras.utils import to_categorical
    from sklearn.preprocessing import LabelEncoder
    y = df.iloc[:,1].values
    print("before: \n",y)
    le=LabelEncoder()
    y=le.fit_transform(y)

    y=to_categorical(y)
    print("After: \n",y)
```

```
before:
 ['INFJ' 'ENTP' 'INTP' ... 'INTP' 'INFP' 'INFP']
After:
 [[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.]]
```

5. Encoding and padding the posts data

This step is necessary to make the posts data to be passed to the model, this step consists of many sub steps which are - porter stemming -> converting the result into a corpus -> encoding the corpus using oneHotEncoder -> applying padding sequences.

```
[ ] from tensorflow.keras.preprocessing.text import one_hot
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    from nltk.stem.porter import PorterStemmer
    from nltk.corpus import stopwords
    nltk.download('stopwords')
    ps=PorterStemmer()
    corpus=[]
    for i in range (0,len(messages)):
        review = re.sub('[^a-zA-Z]','',messages[i])
        review=review.lower()
        review=review.split()

        review = [ps.stem(word) for word in review if not word in stopwords.words('english') ]
        review=''.join(review)
        corpus.append(review)

    oe=[one_hot(words,voc_size) for words in corpus ]
    sent_length=250
    embedded_docs = pad_sequences(oe,padding='pre',maxlen = sent_length)
    embedded_docs
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
array([[    0,     0,     0, ...,     0,     0, 27799],
       [    0,     0,     0, ...,     0,     0, 33768],
       [    0,     0,     0, ...,     0,     0, 30940],
       ...,
       [    0,     0,     0, ...,     0,     0, 47135],
       [    0,     0,     0, ...,     0,     0, 27522],
       [    0,     0,     0, ...,     0,     0, 24640]], dtype=int32)
```

6. Splitting the training and testing data

It is necessary to split our data into training and testing set to ensure that the model is just as effective in the set of data it hasn't seen before as it is on the data that it has trained on to avoid overfitting and underfitting.

```
[ ] from imblearn.over_sampling import RandomOverSampler
    from sklearn.model_selection import train_test_split
    import numpy as np

    X=np.array(embedded_docs)
    ros = RandomOverSampler(random_state=42) # fit predictor and target variable
    x_rus, y_rus = ros.fit_resample(X, y)
    x_train,x_test,y_train,y_test=train_test_split(x_rus, y_rus, test_size=0.2, random_state=42)
```

7. Adding callbacks

We add a few tensorflow callbacks while training our model to avoid overfitting and saving the best fitting model even if it occurred a few epochs back.

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
callbacks = [EarlyStopping( monitor="val_accuracy", patience=2 ),ModelCheckpoint('./model0.h5', monit
```

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
callbacks = [EarlyStopping( monitor="val_accuracy", patience=2 ),ModelCheckpoint('model2.h5', monitor
```

```
callbacks = [EarlyStopping( monitor="val_accuracy", patience=2 ),
             ModelCheckpoint('./model3.h5', monitor = 'val_accuracy', mode = 'max', verbose = 1, save_bes
```

8. Implementation of Model-1 : LSTM

LSTM is a recurrent neural network (RNN) architecture that REMEMBERS values over arbitrary intervals. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. Relative insensitivity to gap length gives an advantage to LSTM over alternative RNNs, hidden Markov models and other sequence learning methods.

```
[ ]
from tensorflow.keras.layers import Embedding,LSTM,Dense,Dropout
dimension=100
model = Sequential()
model.add(Embedding(voc_size,dimension,input_length = sent_length))
model.add(Dropout(0.25))
model.add(LSTM(100))
model.add(Dropout(0.25))
model.add(Dense(16,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer ='adam',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 250, 100)          5000000

 dropout (Dropout)           (None, 250, 100)          0

 lstm (LSTM)                 (None, 100)               80400

 dropout_1 (Dropout)         (None, 100)               0

 dense (Dense)               (None, 16)                1616

=================================================================
Total params: 5,082,016
Trainable params: 5,082,016
Non-trainable params: 0
_________________________________________________________________
```

```python
history = model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=5,batch_size=64, callbacks = callbacks )
```

```
Epoch 1/5
367/367 [==============================] - ETA: 0s - loss: 2.1841 - accuracy: 0.3511
Epoch 1: val_accuracy improved from -inf to 0.75797, saving model to ./model0.h5
367/367 [==============================] - 219s 585ms/step - loss: 2.1841 - accuracy: 0.3511 - val_loss: 0.9889 - val_accuracy: 0.7580
Epoch 2/5
367/367 [==============================] - ETA: 0s - loss: 0.6064 - accuracy: 0.8473
Epoch 2: val_accuracy improved from 0.75797 to 0.84752, saving model to ./model0.h5
367/367 [==============================] - 214s 583ms/step - loss: 0.6064 - accuracy: 0.8473 - val_loss: 0.5023 - val_accuracy: 0.8475
Epoch 3/5
367/367 [==============================] - ETA: 0s - loss: 0.2360 - accuracy: 0.9537
Epoch 3: val_accuracy improved from 0.84752 to 0.85519, saving model to ./model0.h5
367/367 [==============================] - 214s 583ms/step - loss: 0.2360 - accuracy: 0.9537 - val_loss: 0.4967 - val_accuracy: 0.8552
Epoch 4/5
367/367 [==============================] - ETA: 0s - loss: 0.1504 - accuracy: 0.9607
Epoch 4: val_accuracy did not improve from 0.85519
367/367 [==============================] - 213s 579ms/step - loss: 0.1504 - accuracy: 0.9607 - val_loss: 0.4923 - val_accuracy: 0.8502
Epoch 5/5
367/367 [==============================] - ETA: 0s - loss: 0.1312 - accuracy: 0.9593
Epoch 5: val_accuracy did not improve from 0.85519
367/367 [==============================] - 213s 581ms/step - loss: 0.1312 - accuracy: 0.9593 - val_loss: 0.4989 - val_accuracy: 0.8511
```

9.Implementation of Model-2 : Single Layer 1D convolution

In a 1D Convolution model, a single kernel will move one-by-one down a list of input embeddings, looking at the first word embedding (and a small window of next-word embeddings) then the next word embedding, and the next, and so on. The resultant output will be a feature vector that contains about as many values as there were input embeddings. This helps us find patterns in the texts and classify them.

```python
max_features =50000
embedding_dim =64
sequence_length = 250

model2 = tf.keras.Sequential()

model2.add(tf.keras.layers.Embedding(max_features +1, embedding_dim, input_length=sequence_length, ))

model2.add(tf.keras.layers.Conv1D(128,16, activation='relu'))

model2.add(tf.keras.layers.GlobalMaxPooling1D())

model2.add(tf.keras.layers.Dropout(0.5))

model2.add(tf.keras.layers.Dense(16, activation='sigmoid'))

model2.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True), optimizer='Nadam', metrics=["accuracy"])
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 250, 64)           3200064

 conv1d (Conv1D)             (None, 235, 128)          131200

 global_max_pooling1d (Globa  (None, 128)              0
 lMaxPooling1D)

 dropout_2 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 16)                2064

=================================================================
Total params: 3,333,328
Trainable params: 3,333,328
Non-trainable params: 0
```

```python
history_2 = model2.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=5,batch_size=64, callbacks = callbacks )
```

```
Epoch 1/5
367/367 [==============================] - ETA: 0s - loss: 0.2018 - accuracy: 0.9549
Epoch 1: val_accuracy improved from -inf to 0.85946, saving model to model2.h5
367/367 [==============================] - 150s 408ms/step - loss: 0.2018 - accuracy: 0.9549 - val_loss: 0.5217 - val_accuracy: 0.8595
Epoch 2/5
367/367 [==============================] - ETA: 0s - loss: 0.1761 - accuracy: 0.9574
Epoch 2: val_accuracy did not improve from 0.85946
367/367 [==============================] - 141s 385ms/step - loss: 0.1761 - accuracy: 0.9574 - val_loss: 0.5242 - val_accuracy: 0.8591
Epoch 3/5
367/367 [==============================] - ETA: 0s - loss: 0.1622 - accuracy: 0.9587
Epoch 3: val_accuracy improved from 0.85946 to 0.86116, saving model to model2.h5
367/367 [==============================] - 139s 379ms/step - loss: 0.1622 - accuracy: 0.9587 - val_loss: 0.5496 - val_accuracy: 0.8612
Epoch 4/5
367/367 [==============================] - ETA: 0s - loss: 0.1540 - accuracy: 0.9571
Epoch 4: val_accuracy did not improve from 0.86116
367/367 [==============================] - 141s 384ms/step - loss: 0.1540 - accuracy: 0.9571 - val_loss: 0.5220 - val_accuracy: 0.8603
Epoch 5/5
367/367 [==============================] - ETA: 0s - loss: 0.1439 - accuracy: 0.9595
Epoch 5: val_accuracy did not improve from 0.86116
367/367 [==============================] - 144s 393ms/step - loss: 0.1439 - accuracy: 0.9595 - val_loss: 0.5314 - val_accuracy: 0.8596
```

10. Implementation of Model-3 : Double Layer 1D convolution

This works similar to a one layer 1D Convolution network , but as it has two layers the neural network formed by this model is more complex and dense , the 1D Convlayers are followed by MaxPooling layers which help the model group similarities together to avoid differentiating too much between similar terms.

```
max_features =50000
embedding_dim =64
sequence_length = 250

model3 = tf.keras.Sequential()

model3.add(tf.keras.layers.Embedding(max_features +1, embedding_dim, input_length=sequence_length, ))

model3.add(tf.keras.layers.Conv1D(128,16, activation='relu'))

model3.add(tf.keras.layers.MaxPooling1D())

model3.add(tf.keras.layers.Conv1D(128,16, activation='relu'))

model3.add(tf.keras.layers.GlobalMaxPooling1D())

model3.add(tf.keras.layers.Dropout(0.5))

model3.add(tf.keras.layers.Dense(16, activation='sigmoid'))

model3.compile(loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True), optimizer='Nadam', metrics=["accuracy"])
model3.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 250, 64)           3200064

 conv1d_7 (Conv1D)           (None, 235, 128)          131200

 max_pooling1d_3 (MaxPooling  (None, 117, 128)         0
 1D)

 conv1d_8 (Conv1D)           (None, 102, 128)          262272

 global_max_pooling1d_4 (Glo  (None, 128)              0
 balMaxPooling1D)

 dropout_6 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 16)                2064

=================================================================
Total params: 3,595,600
Trainable params: 3,595,600
Non-trainable params: 0
_________________________________________________________________
```

```
history_3 = model3.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=5,batch_size=64, callbacks = callbacks )
```

```
Epoch 1/5
/usr/local/lib/python3.9/dist-packages/keras/backend.py:5561: UserWarning: "`categorical_crossentropy` received `from_log
  output, from_logits = _get_logits(
367/367 [==============================] - ETA: 0s - loss: 2.7730 - accuracy: 0.0605
Epoch 1: val_accuracy improved from -inf to 0.05867, saving model to ./model3.h5
367/367 [==============================] - 253s 684ms/step - loss: 2.7730 - accuracy: 0.0605 - val_loss: 2.7728 - val_acc
Epoch 2/5
367/367 [==============================] - ETA: 0s - loss: 2.7728 - accuracy: 0.0612
Epoch 2: val_accuracy improved from 0.05867 to 0.05987, saving model to ./model3.h5
367/367 [==============================] - 249s 678ms/step - loss: 2.7728 - accuracy: 0.0612 - val_loss: 2.7730 - val_acc
Epoch 3/5
367/367 [==============================] - ETA: 0s - loss: 2.7727 - accuracy: 0.0606
Epoch 3: val_accuracy did not improve from 0.05987
367/367 [==============================] - 248s 675ms/step - loss: 2.7727 - accuracy: 0.0606 - val_loss: 2.7731 - val_acc
Epoch 4/5
367/367 [==============================] - ETA: 0s - loss: 2.7727 - accuracy: 0.0609
Epoch 4: val_accuracy did not improve from 0.05987
367/367 [==============================] - 240s 654ms/step - loss: 2.7727 - accuracy: 0.0609 - val_loss: 2.7732 - val_acc
```

Result Analysis

As we have successfully implemented our three models, let us compare the results:

1. Results of Model-1 : LSTM

```
[ ] model = tf.keras.models.load_model('model0.h5')
    preds = model.predict(x_test)
    eval = model.evaluate(x_test,y_test)
    print("Val. Loss: ",eval[0])
    print("Val. Accuracy: ",eval[1])
```

```
184/184 [==============================] - 8s 42ms/step
184/184 [==============================] - 6s 34ms/step - loss: 0.5018 - accuracy: 0.8596
Val. Loss:  0.5018453001976013
Val. Accuracy:  0.8596282005310059
```

```
from sklearn.metrics import confusion_matrix
true_cat = []
for y in y_test:
  true_cat.append(np.where(y==1)[0])
predicted_cat = tf.argmax(preds, axis=1)
predicted_cat
print(confusion_matrix(predicted_cat, true_cat))
```

```
[[365   0   0   5   0   0   0   0   3   2   3   2   0   0   0   0]
 [  0 309   0   2   0   0   0   0   6   7   6   8   0   2   0   0]
 [  1   1 396   0   0   0   0   0   5   2   1   2   0   2   0   1]
 [  2   4   0 284   0   0   0   0   4   4   7   4   0   5   2   0]
 [  0   0   0   0 382   0   0   0   0   0   2   1   0   0   0   0]
 [  0   0   2   1   0 371   0   0   0   0   0   0   0   0   0   2]
 [  0   0   0   0   0   0 351   0   0   0   0   0   0   2   0   0]
 [  0   0   0   0   0   0   0 391   0   0   0   1   0   0   0   0]
 [  0   3   0   2   0   0   0   0 104   7   9   2   0   0   0   2]
 [  0  36   0  36   0   0   0   0 213 297 128 157   0   2   0   5]
 [  3   1   0   6   0   0   0   0   8   7 212   5   0   0   0   3]
 [  0   0   0   3   0   0   0   0   8  10   2 150   0   0   0   0]
 [  0   0   1   1   0   0   0   0   2   2   1   1 350   0   0   0]
 [  2   1   0   0   0   0   0   0   2   3   4   5   0 371   0   0]
 [  1   0   2   3   0   0   0   0   3   4   2   0   0   0 351   0]
 [  2   5   0   1   0   0   0   0   2   0   2   6   0   3   0 356]]
```

As we can see the LSTM model gives us an validation accuracy of 85.9% , validation loss of 0.5.

2. Results of Model-2 : Single Layer 1D convolution

```python
model = tf.keras.models.load_model('model2.h5')
preds = model.predict(x_test)
eval = model.evaluate(x_test,y_test)
print("Val. Loss: ",eval[0])
print("Val. Accuracy: ",eval[1])
```

```
184/184 [==============================] - 8s 41ms/step
  1/184 [..............................] - ETA: 35s - loss: 0.5051 - accuracy: 0.9062/usr/local,
  output, from_logits = _get_logits(
184/184 [==============================] - 7s 35ms/step - loss: 0.5496 - accuracy: 0.8612
Val. Loss:  0.5495995879173279
Val. Accuracy:  0.8611631989479065
```

```python
from sklearn.metrics import confusion_matrix
true_cat = []
for y in y_test:
  true_cat.append(np.where(y==1)[0])
predicted_cat = tf.argmax(preds, axis=1)
predicted_cat
print(confusion_matrix(predicted_cat, true_cat))
```

```
[[370   0   0   5   0   0   0   0   3   2   3   2   0   0   0   2]
 [  0 308   0   3   0   0   0   0   7   5   6   8   0   2   0   0]
 [  1   1 394   0   0   0   0   0   5   2   1   2   0   2   0   1]
 [  0   5   0 288   0   0   0   0   4   4   7   5   0   0   0   0]
 [  0   0   0   0 382   0   0   0   0   0   2   1   0   0   0   0]
 [  0   0   2   1   0 371   0   0   0   0   0   0   0   0   0   2]
 [  0   0   0   0   0   0 351   0   0   0   0   0   0   2   0   0]
 [  0   0   0   0   0   0   0 391   0   0   0   1   0   0   0   0]
 [  0   3   0   4   0   0   0   0 108   7  11   3   0   0   0   2]
 [  0  38   0  34   0   0   0   0 213 302 131 159   0   4   0   6]
 [  0   1   0   2   0   0   0   0   5   8 205   5   0   0   0   0]
 [  0   0   0   1   0   0   0   0   6   5   3 146   0   0   0   0]
 [  0   0   1   1   0   0   0   0   2   2   1   1 350   0   0   0]
 [  2   1   0   1   0   0   0   0   2   3   4   5   0 374   0   0]
 [  1   0   4   3   0   0   0   0   3   4   2   0   0   0 353   0]
 [  2   3   0   1   0   0   0   0   2   1   3   6   0   3   0 356]]
```

As we can see the single layer 1D convolution model gives us a slightly better result with 86.1% validation accuracy and 0.54 validation loss.

3. Results of Model-3 : Double Layer 1D convolution

```
[ ] model = tf.keras.models.load_model('model3.h5')
    preds = model.predict(x_test)
    eval = model.evaluate(x_test,y_test)
    print("Val. Loss: ",eval[0])
    print("Val. Accuracy: ",eval[1])
```

```
184/184 [==============================] - 15s 81ms/step
184/184 [==============================] - 13s 70ms/step - loss: 2.7730 - accuracy: 0.0599
Val. Loss:  2.7730016708374023
Val. Accuracy:  0.05986696109175682
```

```
[ ] from sklearn.metrics import confusion_matrix
    true_cat = []
    for y in y_test:
      true_cat.append(np.where(y==1)[0])
    predicted_cat = tf.argmax(preds, axis=1)
    predicted_cat
    print(confusion_matrix(predicted_cat, true_cat))
```

```
[[  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   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   0   0   0   0   0   0   0   0   0   0]
 [376 360 401 344 382 371 351 391 360 345 379 344 350 387 353 369]
 [  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   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   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   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

As we can see the model 3 just gives us an validation accuracy of 5.9% and a validation loss of 2.77 even though this model is more complex than our Model 1 and 2, this is due to imbalance in classes i.e as seen in the data visualisation section that the INFP class has 1832records while the ESTJ class had only 39 records , therefore for a dataset of this sort a simpler model performs better.

The best performing model was the Model-2 which is the single layer 1D convolution modelwith an accuracy of 86.1%.

All our models were able to avoid overfitting and reduce execution time by terminating the training when there was no improvement in the model recorded after a few epochs.

We were able to save the best fit for each of the models as model0.h5, model2.h5 and model3.h5 files. With this we were able to train a deep learning model which predicts a person’s personality type using their social media posts.

References-

Sun, J., Tian, Z., Fu, Y., Geng, J., & Liu, C. (2021). Digital twins in human understanding: a deep learning-based method to recognize personality traits. *International Journal of Computer Integrated Manufacturing*, *34*(7-8), 860-873.

Deilami, F. M., Sadr, H., & Nazari, M. (2022). Using machine learning based models for personality recognition. *arXiv preprint arXiv:2201.06248*.

Vinciarelli, A., & Mohammadi, G. (2014). A survey of personality computing. *IEEE Transactions on Affective Computing*, *5*(3), 273-291.

Majumder, N., Poria, S., Gelbukh, A., & Cambria, E. (2017). Deep learning-based document modeling for personality detection from text. *IEEE Intelligent Systems*, *32*(2), 74-79.

Cambria, E., Das, D., Bandyopadhyay, S., & Feraco, A. (2017). Affective computing and sentiment analysis. *A practical guide to sentiment analysis*, 1-10.

Yu, J., & Markov, K. (2017, November). Deep learning based personality recognition from facebook status updates. In *2017 IEEE 8th international conference on awareness science and technology (iCAST)* (pp. 383-387). IEEE.

Ren, Z., Shen, Q., Diao, X., & Xu, H. (2021). A sentiment-aware deep learning approach for personality detection from text. *Information Processing & Management*, *58*(3), 102532.

Carducci, G. (2018). *Semantic Analysis to Compute Personality Traits from Social Media Posts* (Doctoral dissertation, Politecnico di Torino).

Xue, D., Hong, Z., Guo, S., Gao, L., Wu, L., Zheng, J., & Zhao, N. (2017). Personality recognition on social media with label distribution learning. *IEEE access*, *5*, 13478-13488.

Samani, Z. R., Guntuku, S. C., Moghaddam, M. E., Preoţiuc-Pietro, D., & Ungar, L. H. (2018). Cross-platform and cross-interaction study of user personality based on images on Twitter and Flickr. *PloS one*, *13*(7), e0198660.