# Text Classification:

## Data

1. we have total of 20 types of documents(Text files) and total 18828 documents(text files).
2. You can download data from this link, in that you will get documents.rar folder.
If you unzip that, you will get total of 18828 documnets. document name is defined as'ClassLabel_DocumentNumberInThatLabel'.
so from document name, you can extract the label for that document.
4. Now our problem is to classify all the documents into any one of the class.
5. Below we provided count plot of all the labels in our data.

In [ ]:

```
!pip install ipython-autotime
%load_ext autotime
```

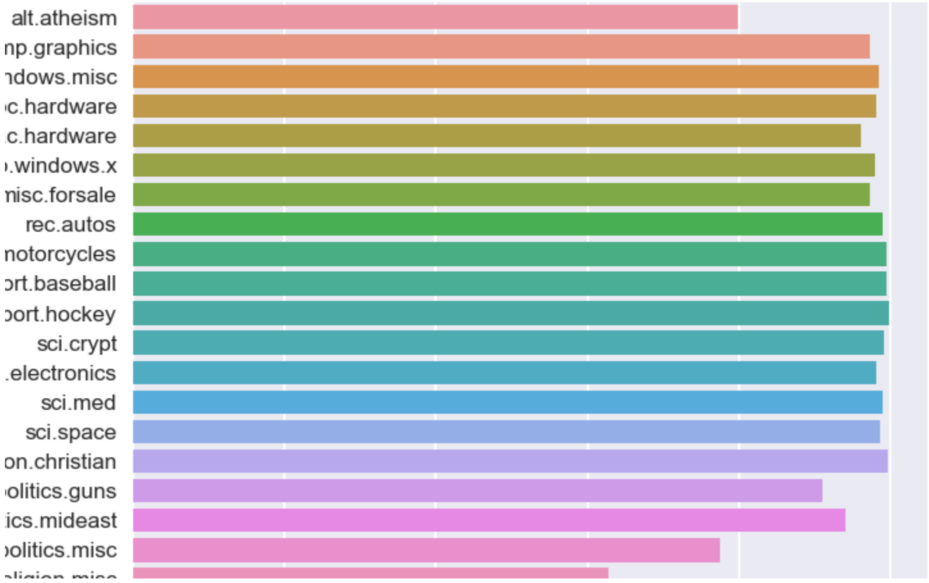
```
Collecting ipython-autotime
  Downloading
https://files.pythonhosted.org/packages/e6/f9/0626bbdb322e3a078d968e87e3b01341e7890544de891d0cb6136
0e6/ipython-autotime-0.1.tar.bz2
Building wheels for collected packages: ipython-autotime
  Building wheel for ipython-autotime (setup.py) ... done
  Created wheel for ipython-autotime: filename=ipython_autotime-0.1-cp36-none-any.whl size=1831 sh
a256=2a988f2aa89fde4ede696020d9d3d6c210f5e90522a708f2c5bd985e8987399e
  Stored in directory:
/root/.cache/pip/wheels/d2/df/81/2db1e54bc91002cec40334629bc39cfa86dff540b304ebcd6e
Successfully built ipython-autotime
Installing collected packages: ipython-autotime
Successfully installed ipython-autotime-0.1
```

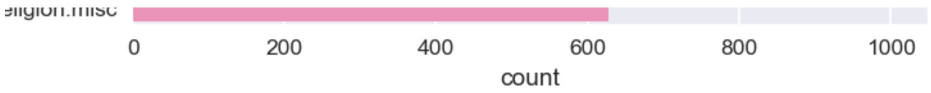
In [ ]:

```
!unrar x documents.rar
```

In [ ]:

```
### count plot of all the class labels.
```





In [ ]:

# Assignment:

#### sample document

```
Subject: A word of advice
From: jcopelan@nyx.cs.du.edu (The One and Only)

In article < 65882@mimsy.umd.edu > mangoe@cs.umd.edu (Charley Wingate) writes:
>
>I've said 100 times that there is no "alternative" that should think you
>might have caught on by now.  And there is no "alternative", but the point
>is, "rationality" isn't an alternative either.  The problems of metaphysical
>and religious knowledge are unsolvable-- or I should say, humans cannot
>solve them.

How does that saying go: Those who say it can't be done shouldn't interrupt
those who are doing it.

Jim
--
Have you washed your brain today?
```

## Preprocessing:

```
useful links: http://www.pyregex.com/

1. Find all emails in the document and then get the text after the "@". and then split thos
e texts by '.'
after that remove the words whose length is less than or equal to 2 and also remove'com' wo
rd and then combine those words by space.
In one doc, if we have 2 or more mails, get all.
Eg:[test@dm1.d.com, test2@dm2.dm3.com]-->[dm1.d.com, dm3.dm4.com]-->[dm1,d,com,dm2,dm3,com]-
->[dm1,dm2,dm3]-->"dm1 dm2 dm3"
append all those into one list/array. ( This will give length of 18828 sentences i.e one li
st for each of the document).
Some sample output was shown below.

> In the above sample document there are emails [jcopelan@nyx.cs.du.edu,
65882@mimsy.umd.edu, mangoe@cs.umd.edu]

preprocessing:
[jcopelan@nyx.cs.du.edu, 65882@mimsy.umd.edu, mangoe@cs.umd.edu] ==> [nyx cs du edu mimsy
umd edu cs umd edu] ==>
[nyx edu mimsy umd edu umd edu]

2. Replace all the emails by space in the original text.
```

In [ ]:

In [ ]:

```
len(preprocessed_email)
```

Out[ ]:

```
18828
```

3. Get subject of the text i.e. get the total lines where "Subject:" occur and remove the word which are before the ":" remove the newlines, tabs, punctuations, any special chars.
**Eg: if we have sentance like "Subject: Re: Gospel Dating @ \r\r\n" --> You have to get "Gospel Dating"**
Save all this data into another list/array.

4. After you store it in the list, Replace those sentances in original text by space.

5. Delete all the sentances where sentence starts with **"Write to:"** or **"From:"**.
> In the above sample document check the 2nd line, we should remove that

6. Delete all the tags like "< anyword >"
> In the above sample document check the 4nd line, we should remove that "< 65882@mimsy.umd.edu >"

7. Delete all the data which are present in the brackets.
In many text data, we observed that, they maintained the explanation of sentence or translation of sentence to another language in brackets so remove all those.
**Eg: "AAIC-The course that gets you HIRED(AAIC - Der Kurs, der Sie anstellt)" --> "AAIC-The course that gets you HIRED"**

> In the above sample document check the 4nd line, we should remove that "(Charley Wingate)"

8. Remove all the newlines('\n'), tabs('\t'), "-", "\".

9. Remove all the words which ends with ":".
**Eg: "Anyword:"**
> In the above sample document check the 4nd line, we should remove that "writes:"

10. Decontractions, replace words like below to full words.
please check the donors choose preprocessing for this
**Eg: can't -> can not, 's -> is, i've -> i have, i'm -> i am, you're -> you are, i'll --> i will**

**There is no order to do point 6 to 10. but you have to get final output correctly**

11. Do chunking on the text you have after above preprocessing.
Text chunking, also referred to as shallow parsing, is a task that
follows Part-Of-Speech Tagging and that adds more structure to the sentence.
So it combines the some phrases, named entities into single word.
So after that combine all those phrases/named entities by separating "_".
And remove the phrases/named entities if that is a "Person".
You can use **nltk.ne_chunk** to get these.
Below we have given one example. please go through it.

useful links:
https://www.nltk.org/book/ch07.html
https://stackoverflow.com/a/31837224/4084039
http://www.nltk.org/howto/tree.html
https://stackoverflow.com/a/44294377/4084039

In [ ]:

```
#i am living in the New York
print("i am living in the New York -->", list(chunks))
print(" ")
print("-"*50)
print(" ")
#My name is Srikanth Varma
print("My name is Srikanth Varma -->", list(chunks1))
```

```
i am living in the New York --> [('i', 'NN'), ('am', 'VBP'), ('living', 'VBG'), ('in', 'IN'), ('th
e', 'DT'), Tree('GPE', [('New', 'NNP'), ('York', 'NNP')])]

--------------------------------------------------

My name is Srikanth Varma --> [('My', 'PRP$'), ('name', 'NN'), ('is', 'VBZ'), Tree('PERSON', [('Sr
ikanth', 'NNP'), ('Varma', 'NNP')])]
```

We did chunking for above two lines and then We got one list where each word is mapped to a

POS(parts of speech) and also if you see "New York" and "Srikanth Varma",
they got combined and represented as a tree and "New York" was referred as "GPE" and "Srikanth Varma" was referred as "PERSON".
so now you have to Combine the "New York" with "_" i.e "New_York"
and remove the "Srikanth Varma" from the above sentence because it is a person.

**13.** Replace all the digits with space i.e delete all the digits.
> In the above sample document, the 6th line have digit 100, so we have to remove that.

**14.** After doing above points, we observed there might be few word's like
**"_word_" (i.e starting and ending with the _)**, **"_word" (i.e starting with the _)**,
**"word_" (i.e ending with the _)** remove the _ from these type of words.

**15.** We also observed some words like **"OneLetter_word"- eg: d_berlin**,
**"TwoLetters_word" - eg: dr_berlin** , in these words we remove the "OneLetter_" (d_berlin ==> berlin) and
"TwoLetters_" (de_berlin ==> berlin). i.e remove the words
which are length less than or equal to 2 after spliiting those words by "_".

**16.** Convert all the words into lower case and lowe case
and remove the words which are greater than or equal to 15 or less than or equal to 2.

**17.** replace all the words except "A-Za-z_" with space.

**18.** Now You got Preprocessed Text, email, subject. create a dataframe with those.
Below are the columns of the df.

In [ ]:

```
data.columns
```

```
Index(['text', 'class', 'preprocessed_text', 'preprocessed_subject',
       'preprocessed_emails'],
      dtype='object')
```

In [ ]:

```
data.iloc[400]
```

```
text                    From: arc1@ukc.ac.uk (Tony Curtis)\r\r\r\nSubj...
class                                                         alt.atheism
preprocessed_text       said re is article if followed the quoting rig...
preprocessed_subject                                christian morality is
preprocessed_subject                                     christian morality is
preprocessed_emails                                        ukc mac macalstr edu
Name: 567, dtype: object
```

In [ ]:

```
!wget --header="Host: doc-00-bo-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0
(Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.135
Safari/537.36" --header="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/
d-exchange;v=b3;q=0.9" --header="Accept-Language: en-IN,en-GB;q=0.9,en-US;q=0.8,en;q=0.7" --header
="Referer: https://drive.google.com/" --header="Cookie:
AUTH_gnb78hdmdiks9t0b8kec09hpa7nncs5e_nonce=oca5ql1nk257o; _ga=GA1.2.1804417035.1594643089; NID=20
4=WE4tnDNREeWA-
i7VkayXZRT0nZVp1JeYYk6hLWon_UP0rMptB4l1jfZkhNOvbLUPVxDowH066xA42Zz173_rsIlQAnYpv2qrlmQZ9MjqZVljYcd
wd0ZAN7SEaKwZ40sU9zdkP95PVkKfH4uFw5BkGh4qZanzjr9Y-b7iDM" --header="Connection: keep-alive"
"https://doc-00-bo-
docs.googleusercontent.com/docs/securesc/lcn000d4f5ncb3531bgn3uus2eb0i5pv/jde82rd5s13jhfr7ljfpa7ma
6jk/1598576925000/00484516897554883881/03515051603858730688/1rxD15nyeIPIAZ-J2VYPrDRZI66-TBWvM?e=do
wnload&authuser=0&nonce=oca5ql1nk257o&user=03515051603858730688&hash=vf91fl9r1im4hie3mb4ob101msanar
-c -O 'documents.rar'
```

```
--2020-08-28 01:09:22--  https://doc-00-bo-
docs.googleusercontent.com/docs/securesc/lcn000d4f5ncb3531bgn3uus2eb0i5pv/jde82rd5s13jhfr7ljfpa7maj
6jk/1598576925000/00484516897554883881/03515051603858730688/1rxD15nyeIPIAZ-J2VYPrDRZI66-TBWvM?
e=download&authuser=0&nonce=oca5ql1nk257o&user=03515051603858730688&hash=vf91fl9r1im4hie3mb4ob101ms
kr
Resolving doc-00-bo-docs.googleusercontent.com (doc-00-bo-docs.googleusercontent.com)...
74.125.143.132, 2a00:1450:4013:c03::84
Connecting to doc-00-bo-docs.googleusercontent.com (doc-00-bo-
docs.googleusercontent.com)|74.125.143.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/rar]
Saving to: ‘documents.rar’

documents.rar           [  <=>               ]  18.16M  87.1MB/s    in 0.2s

2020-08-28 01:09:23 (87.1 MB/s) - ‘documents.rar’ saved [19038123]
```

### To get above mentioned data frame --> Try to Write Total Preprocessing steps in One Function Named Preprocess as below.

In [ ]:

```python
import re
import nltk
from chardet import detect
import numpy as np
import pandas as pd

nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')

def preprocess_sub(Input_Text):
    """Do all the Preprocessing as shown above and
    return a tuple contain preprocess_email,preprocess_subject,preprocess_text for that
Text_data"""
    text_list=[]
    def get_encoding_type(file):
      with open(file, 'rb') as f:
          rawdata = f.read()
      return detect(rawdata)['encoding']

    def underscore(input):
      len_check=0
      input_list=input.split(" ")
      for i,e in enumerate(input_list):
          if '_' in e:
            sublist=e.split("_")
            if ( all(len(i) >= 2 for i in sublist)):
              len_check=1
            for j,k in enumerate(sublist):
              if (len(k)<3):
                len_check=0
                sublist[j]=""
            if (len_check==1):
              sublist='_'.join(sublist)
            else:
              sublist=''.join(sublist)
            input_list[i]=sublist

      input_list=' '.join(input_list)
      return input_list

    def chunking(input):
      parsing_tree = nltk.ne_chunk(nltk.tag.pos_tag(input.split()))   # POS tagging before chunking!
      named_entities = []
      actual_place_words=[]
      final_place_words=[]
      person=[]
      for parse in parsing_tree.subtrees():
        if parse.label() == 'PERSON':
          for index,ele in enumerate(parse):
            person.append(ele[0])

        if parse.label() == 'GPE':
          word=[]
          for index,ele in enumerate(parse):
            word.append(ele[0])
          final_word = '_'.join(word)
          actual_word=' '.join(word)
          actual_place_words.append(actual_word)
          final_place_words.append(final_word)

      sentence_list=input.split(" ")
      for i,e in enumerate(sentence_list):
        if e in person:
          sentence_list[i]=""
      sentence_list=' '.join(sentence_list)
      for i,e in enumerate(actual_place_words):
        sentence_list=sentence_list.split(e)
        sentence_list=final_place_words[i].join(sentence_list)
      return  sentence_list




    def decontracted(phrase):
    # specific
      phrase = re.sub(r"won't", "will not", phrase)
      phrase = re.sub(r"can\'t", "can not", phrase)
      phrase = re.sub(r"can\'t", "can not", phrase)

    # general
      phrase = re.sub(r"n\'t", " not", phrase)
      phrase = re.sub(r"\'re", " are", phrase)
      phrase = re.sub(r"\'s", " is", phrase)
      phrase = re.sub(r"\'d", " would", phrase)
      phrase = re.sub(r"\'ll", " will", phrase)
      phrase = re.sub(r"\'t", " not", phrase)
      phrase = re.sub(r"\'ve", " have", phrase)
      phrase = re.sub(r"\'m", " am", phrase)
      return phrase

    encode_type=get_encoding_type(Input_Text)
    file1 = open(Input_Text,"r",encoding = encode_type, errors='ignore')
    line_remove=False
    whole_text=file1.read()
    text=[]


    # point 1 and 2
    # Eg:[test@dm1.d.com, test2@dm2.dm3.com]-->[dm1.d.com, dm3.dm4.com]-->[dm1,d,com,dm2,dm3,com]->[dm1,dm2,dm3]-->"dm1 dm2 dm3"
    # Replace all the emails by space in the original text.
    x = re.findall(r'@(\w+[\w.]*)', whole_text)
    mail_list=[]
    for i in x:
      ele_list=i.split('.')
      for j in ele_list:
        if len(j)<3:
          ele_list.remove(j)
      mail_list+=ele_list
    mail_string = ' '.join(mail_list)
    mail_string = re.sub(r"com", "", mail_string)
    emails=mail_string


    for line in whole_text.splitlines():
      line_remove=False
      if ("Subject:" in line):
        line_remove=True
        Subject_line=line     #fetching subject line for subject

      # Delete all the sentances where sentence starts with "Write to:" or "From:".
      if ('From:' in line) or ('Write to:' in line): #point 5
        line_remove=True

      if (">" in line) and (len(line)==1):

        line_remove=True
      if(line.strip() == '') or (line == '\n'):

        line_remove=True
      if (line_remove==False):

        # Delete all the tags like "< anyword >"
        # Eg: "AAIC-The course that gets you HIRED(AAIC - Der Kurs, der Sie anstellt)" --> "AAIC-T
he course that gets you HIRED"
        line=re.sub("[\<\(\[].*?[\)\]\>]", "", line)

        #Remove all the newlines('\n'), tabs('\t'), "-", "\".
        line = re.sub('\W+',' ', line )

        #Remove all the words which ends with ":".
        line=re.sub(r'\w+:\s?','',line)

        # Decontractions, replace words like below to full words.
        line=decontracted(line)

        # Do chunking on the text you have after above preprocessing.
        line=chunking(line)

        # Replace all the digits with space i.e delete all the digits.

        line=re.sub('\d', '', line)

        # remove the _ from these type of words.
        # which are length less than or equal to 2 after spliiting those words by "_".
        line=re.sub('_', '', line)
        line=underscore(line)

        # Convert all the words into lower case and lowe case
        line=line.lower()

        # remove the words which are greater than or equal to 15 or less than or equal to 2.
        line=re.sub(r'\b\w{1,2}\b', '', line)
        line=re.sub(r'\b\w{15,50}\b', '', line)

        # replace all the words except "A-Za-z_" with space.
        line = re.sub(r'[^a-zA-Z_]', ' ', line)

        if(len(line)>2):
          text_list.append(line)


    text=' '.join(text_list)
    text=re.sub(' +', ' ', text)  # removing double or triple spaces

    # point 3 and 4
    # Eg: if we have sentance like "Subject: Re: Gospel Dating @ \r\r\n" --> You have to get "Gosp
el Dating"
    # After you store it in the list, Replace those sentances in original text by space.
    subject=[]
    subject_ele = Subject_line.split(':')
    subject_ele=subject_ele[(len(subject_ele)-1)]
    subject_ele = re.sub('\W+',' ', subject_ele )
    subject=subject_ele

    return emails,subject,text
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
time: 1.98 s
```

### Code checking:

After Writing preprocess function. call that functoin with the input text of 'alt.atheism_49960' doc and print the output of the preprocess function
This will help us to evaluate faster, based on the output we can suggest you if there are any changes.

**verification**

In [ ]:

```
#alt.atheism_49960
e,s,t=preprocess_sub("documents/alt.atheism_49960.txt")
print("email:",e)
print("subject:",s)
print("text:",t)
```

```
email: mantis uk net  mantis uk
subject:  Atheist Resources
text: archive name atheism resources alt atheism archive name resources last modified december ver
sion atheist resources addresses atheist organizations usa freedom from religion foundation darwin
fish bumper stickers and assorted other atheist paraphernalia are available from the freedom from
religion foundation the telephone evolution designs designs sell the darwin fish fish symbol like
the ones christians stick their cars but with feet and the word written inside the deluxe moulded
plastic fish postpaid the people the san francisco bay area can get from try mailing for net peopl
e who directly the price per fish american atheist press aap publish various atheist books
critiques the bible lists biblical contradictions and one such book the bible handbook and foote a
merican atheist press isbn edition bible contradictions absurdities atrocities immoralities
contains the bible contradicts based the king version the bible telephone fax prometheus books sel
l books including telephone alternate address prometheus books african americans for humanism
organization promoting black secular humanism and uncovering the history black freethought they pu
blish quarterly newsletter aah examiner united kingdom rationalist press association national secu
lar society street holloway road london london british humanist association south place ethical so
ciety lamb london wcr red lion square london wcr fax the national secular society publish the free
thinker monthly magazine founded germany ibka bund der und postfach berlin germany ibka publish jo
urnal miz materialien und zur zeit politisches journal der und ibka miz berlin germany for atheist
books write ibdk ucherdienst der postfach hannover germany telephone fiction thomas disch the sant
a claus compromise short story the ultimate proof that exists all characters and events are
fictitious any similarity living dead gods well walter canticle for leibowitz one gem this post at
omic doomsday novel the monks who spent their lives copying blueprints from filling the sheets pap
er with ink and leaving white lines and letters edgar pangborn davy post atomic doomsday novel set
clerical states the church for example forbids that anyone produce describe use any substance cont
aining atoms philip dick wrote many philosophical and thought provoking short stories and novels h
is stories are bizarre times but very approachable wrote mainly but wrote about people truth and r
eligion rather than technology although often believed that had met some sort god remained sceptic
al amongst his novels the following are some relevance pot healer fallible alien deity summons gro
up earth craftsmen and women remote planet raise giant cathedral from beneath the oceans when the
deity begins demand faith from the earthers pot healer unable comply polished ironic and amusing n
ovel maze death noteworthy for its description technology based religion valis the schizophrenic h
ero searches for the hidden mysteries gnostic christianity after reality fired into his brain pink
laser beam unknown but possibly divine origin accompanied his dogmatic and dismissively atheist fr
iend and assorted other odd characters the divine invasion god invades making young woman pregnant
she returns from another star system unfortunately she terminally ill and must assisted dead man w
hose brain wired hour easy listening music margaret atwood the handmaid story based the premise th
at the congress mysteriously assassinated and quickly take charge the nation set right again the b
```

```
ook the diary woman life she tries live under the new christian theocracy women right own property
revoked and their bank accounts are closed sinful luxuries are outlawed and the radio only used fo
r readings from the bible crimes are punished retroactively doctors who performed legal abortions
the old world are hunted down and hanged writing style difficult get used first but the tale grows
more and more chilling goes various authors the bible this somewhat dull and rambling work has oft
en been criticized however probably worth reading only that you know what all the fuss about exist
s many different versions make sure you get the one true version non fiction peter rosa vicars chr
ist bantam press although seems even catholic this very enlighting history papal immoralities
adulteries fallacies etc german translation erste dunkle seite des knaur michael martin atheism ph
ilosophical justification temple university press philadelphia usa detailed and scholarly
justification atheism contains outstanding appendix defining terminology and usage this
tendentious area argues both for negative atheism the non belief the existence god and also for po
sitive atheism the belief the non existence god includes great refutations the most challenging ar
guments for god particular attention paid refuting contempory theists such and swinburne pages isb
n the case against christianity temple university press comprehensive critique christianity which
considers the best contemporary defences christianity and demonstrates that they are unsupportable
and incoherent pages isbn turner creed the johns hopkins university press baltimore usa subtitled
the origins unbelief america examines the way which unbelief became mainstream alternative world v
iew focusses the period and while considering france and britain the emphasis american and particu
larly newengland developments neither religious history secularization atheism rather the
intellectual history the fate single idea the belief that exists pages isbn george seldes the grea
t thoughts usa dictionary quotations different kind concentrating statements and writings which ex
plicitly implicitly present the person philosophy and world view includes obscure opinions from ma
ny people for some popular observations traces the way which various people expressed and twisted
the idea over the centuries quite number the quotations are derived from what religion and religio
n pages isbn swinburne the existence god clarendon paperbacks oxford this book the second volume t
rilogy that began with the coherence theism and was concluded with and this work swinburne attempt
s construct series inductive arguments for the existence god his arguments which are somewhat tend
entious and rely upon the imputation late century western christian values and aesthetics god whic
h supposedly simple can conceived were decisively rejected mackie the miracle theism the revised e
dition the existence god swinburne includes appendix which makes somewhat incoherent attempt rebut
mackie the miracle theism oxford this volume contains comprehensive review the principal arguments
for and against the existence god ranges from the classical philosophical positions through the mo
ral arguments newman kant and the recent restatements the classical theses and swinburne also addr
esses those positions which push the concept god beyond the realm the rational such those
kierkegaard kung and well replacements for god such axiarchism the book delight read less
formalistic and better written than works and refreshingly direct when compared with the hand wavi
ng swinburne james haught horrors illustrated history religious murder and madness prometheus book
s looks religious persecution from ancient times the present day and not only christians library c
ongress catalog card number norm allen african american humanism anthology see the listing for afr
ican americans for humanism above gordon stein anthology atheism and anthology covering wide range
subjects including the devil evil and morality and the history freethought comprehensive
bibliography edmund cohen the mind the bible believer prometheus books study why people become chr
istian and what effect has them net resources there small mail based archive server mantis which c
arries archives old alt atheism moderated articles and assorted other files for more information s
end mail archive server mantis saying help send atheism index and will mail back reply mathew
time: 745 ms
```

### After writing Preprocess function, call the function for each of the document(18828 docs) and then create a dataframe as mentioned above.

In [ ]:

```python
import os

def preprocess():
  preprocessed_emails=[]
  preprocessed_subject=[]
  preprocessed_text=[]
  class_ =[]
  files = os.listdir("documents/")
  for file in files:
    if(file.endswith("txt")):
      prefix_file=file.split("_")                    # Spilt the file based "_" ( Ex: File name : misc.f
orsale_76227.txt , prefix_file[0] = misc.forsale and prefix_file[1] = 76227.txt  )
      class_list =prefix_file[0]
      class_.append(class_list)
      file_path="documents/"+file
      mail,sub,text=preprocess_sub(file_path)
      preprocessed_emails.append(mail)
      preprocessed_subject.append(sub)
      preprocessed_text.append(text)

  mail_ = pd.DataFrame(preprocessed_emails)
  subject_ = pd.DataFrame(preprocessed_subject)
  subject_ = pd.DataFrame(preprocessed_subject)
  class_data = pd.DataFrame(class_)
  text_ = pd.DataFrame(preprocessed_text)
  df=pd.DataFrame(np.column_stack([mail_, subject_, text_,class_data]),
columns=['preprocessed_email','preprocessed_subject','preprocessed_text','text_class'])
  return df

data =preprocess()

# Print shape of the data frame 18828 X 4
data.shape
```

Out[ ]:

```
(18828, 4)
```

```
time: 21min 41s
```

**DataFrame**

## Training The models to Classify:

1. Combine "preprocessed_text", "preprocessed_subject", "preprocessed_emails" into one column. use that column to model.

2. Now Split the data into Train and test. use 25% for test also do a stratify split.

3. Analyze your text data and pad the sequnce if required.
Sequnce length is not restricted, you can use anything of your choice.
you need to give the reasoning

4. Do Tokenizer i.e convert text into numbers. please be careful while doing it.
if you are using tf.keras "Tokenizer" API, it removes the **"_"**, but we need that.

5. code the model's ( Model-1, Model-2 ) as discussed below
and try to optimize that models.

6. For every model use predefined Glove vectors.
**Don't train any word vectors while Training the model.**

7. Use "categorical_crossentropy" as Loss.

8. Use **Accuracy and Micro Avgeraged F1 score** as your as Key metrics to evaluate your model.

9.  Use Tensorboard to plot the loss and Metrics based on the epoches.

10. Please save your best model weights in to **'best_model_L.h5' ( L = 1 or 2 )**.

11. You are free to choose any Activation function, learning rate, optimizer.
But have to use the same architecture which we are giving below.

12. You can add some layer to our architecture but you **deletion** of layer is not acceptable.

13. Try to use **Early Stopping** technique or any of the callback techniques that you did in the previous assignments.

14. For Every model save your model to image ( Plot the model) with shapes
and inlcude those images in the notebook markdown cell,
upload those imgages to Classroom. You can use "plot_model"
please refer this if you don't know how to plot the model with shapes.

In [ ]:

```
data.head()
```

Out[ ]:

| | preprocessed_email | preprocessed_subject | preprocessed_text | text_class |
|---|---|---|---|---|
| **0** | grover stat washington edu | Wierd xdm behavior | dear sun and windows people running sun workst... | comp.windows.x |
| **1** | cutting hou us | Ford SHO engine parts | will not work internal engine components the ... | rec.autos |
| **2** | rins ryukoku jp phantom gatech edu phantom gat... | nuclear waste | article matthew phantom gatech edu writes gre... | sci.space |
| **3** | westminster uk | Dirty Diesels | yeah diesels are cleaner than petrol powered c... | rec.autos |
| **4** | leo bsuvc bsu edu camelot camelot bradley edu | CUB fever | article kingoz camelot bradley edu writes cub... | rec.sport.baseball |

```
time: 34 ms
```

In [ ]:

```python
# Combine "preprocessed_text", "preprocessed_subject", "preprocessed_emails" into one column. use 
that column to model.
data["data"]=data["preprocessed_email"] +" "+ data["preprocessed_subject"]+"
"+data["preprocessed_text"]
max_length=0
for i in data["data"]:
  if (len(i)> max_length):
    max_length=len(i)
print("max_length:",max_length)
```

```
max_length: 56307
time: 70.6 ms
```

In [ ]:

```python
data["text_class"].shape
```

Out[ ]:

```
(18828,)
```

```
time: 2.67 ms
```

Splitting Data:

In [ ]:

```python
# Now Split the data into Train and test. use 25% for test also do a stratify split.

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data["data"], data["text_class"],stratify=data[
"text_class"],test_size=0.25)
```

```
time: 31.8 ms
```

In [ ]:

```python
print(X_train.shape)
print(y_train.shape)
```

```
(14121,)
(14121,)
time: 1.22 ms
```

In [ ]:

```python
import tensorflow as tf
import keras
```

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

```
time: 1.3 s
```

Tokenizing using Keras API:

In [ ]:

```
# Do Tokenizer i.e convert text into numbers. please be careful while doing it. if you are using t
f.keras "Tokenizer" API, it removes the "_", but we need that.
token_  = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n')
token_.fit_on_texts(X_train)
X_train = token_.texts_to_sequences(X_train)
X_test = token_.texts_to_sequences(X_test)

print(len(X_train))
print(len(X_test))
```

```
14121
4707
time: 3.1 s
```

labeling classes from string to interger and then encoding them:

In [ ]:

```
from sklearn.preprocessing import LabelEncoder

def prepare_targets(y_train, y_test):
 le = LabelEncoder()
 le.fit(y_train)
 y_train = le.transform(y_train)
 y_test = le.transform(y_test)
 return y_train, y_test

y_train, y_test=prepare_targets(y_train, y_test)
```

```
time: 9.24 ms
```

In [ ]:

```
y_train = keras.utils.to_categorical(y_train, num_classes=20)
y_test = keras.utils.to_categorical(y_test, num_classes=20)
```

```
time: 3.35 ms
```

In [ ]:

```
# Analyze your text data and pad the sequnce if required. Sequnce length is not restricted, you ca
n use anything of your choice. you need to give the reasoning
# Padding the data,in order to bring stability and inprove processing for batches:
X_train = pad_sequences(X_train, maxlen=max_length, padding='post')
X_test = pad_sequences(X_test, maxlen=max_length, padding='post')
```

```
time: 1.31 s
```

In [ ]:

```
def load_embedding(filename,encoding):
    file = open(filename,'r',encoding=encoding)
    lines = file.readlines()[1:]
    file.close()
    embedding = dict()
    for line in lines:
        parts = line.split()
        embedding[parts[0]] = asarray(parts[1:], dtype='float32')
        embedding[parts[0]] = asarray(parts[1:], dtype='float32')
    return embedding
```

```
time: 2.42 ms
```

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense,Dropout,Input
from keras.layers import Flatten,BatchNormalization
from keras.layers import Embedding
from keras.layers import Concatenate
from keras.models import Model
from keras.optimizers import Adam
from keras import regularizers


from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
```

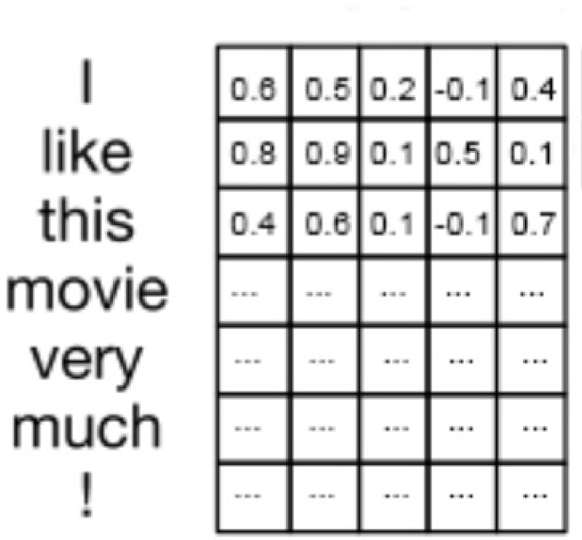
```
time: 3.97 ms
```

Keras prefers inputs to be vectorized and all inputs to have the same length

# reference:https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html

### Model-1: Using 1D convolutions with word embeddings

**Encoding of the Text** --> For a given text data create a Matrix with Embedding layer as shown Below.
In the example we have considered d = 5, but in this assignment we will get d = dimension of Word vectors we are using.
i.e if we have maximum of 350 words in a sentence and embedding of 300 dim word vector,
we result in 350*300 dimensional matrix for each sentance as output after embedding layer



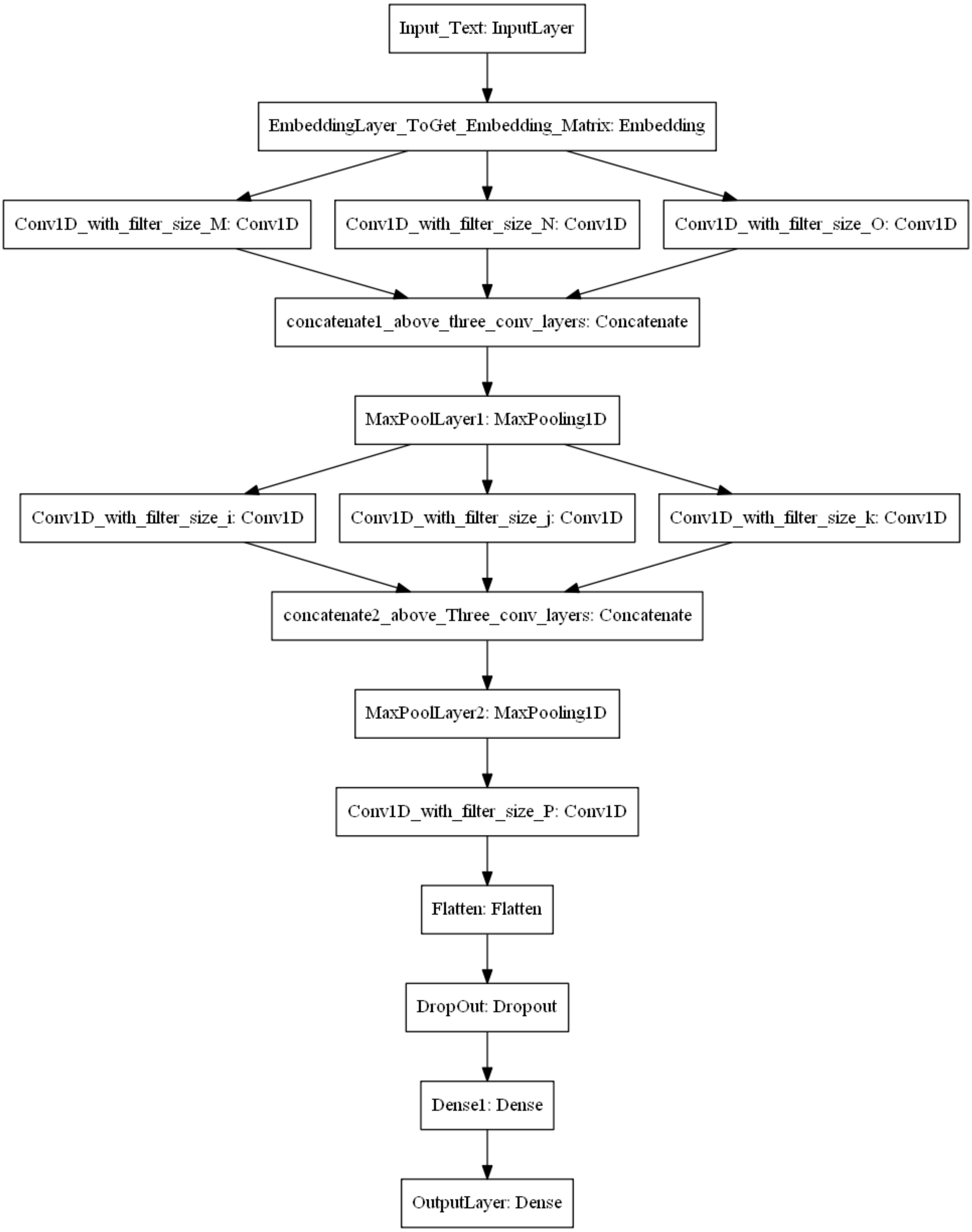
| | | | | | |
|---|---|---|---|---|---|
| I | 0.6 | 0.5 | 0.2 | -0.1 | 0.4 |
| like | 0.8 | 0.9 | 0.1 | 0.5 | 0.1 |
| this | 0.4 | 0.6 | 0.1 | -0.1 | 0.7 |
| movie | ... | ... | ... | ... | ... |
| very | ... | ... | ... | ... | ... |
| much | ... | ... | ... | ... | ... |
| ! | ... | ... | ... | ... | ... |

Ref: https://i.imgur.com/kiVQuk1.png

**Reference:**
https://stackoverflow.com/a/43399308/4084039
https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-networks-keras/

How EMBEDDING LAYER WORKS

**Go through this blog, if you have any doubt on using predefined Embedding values in Embedding layer - https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/**



Input_Text: InputLayer
EmbeddingLayer_ToGet_Embedding_Matrix: Embedding
Conv1D_with_filter_size_M: Conv1D
Conv1D_with_filter_size_N: Conv1D
Conv1D_with_filter_size_O: Conv1D
concatenate1_above_three_conv_layers: Concatenate
MaxPoolLayer1: MaxPooling1D
Conv1D_with_filter_size_i: Conv1D
Conv1D_with_filter_size_j: Conv1D
Conv1D_with_filter_size_k: Conv1D
concatenate2_above_Three_conv_layers: Concatenate
MaxPoolLayer2: MaxPooling1D
Conv1D_with_filter_size_P: Conv1D
Flatten: Flatten
DropOut: Dropout
Dense1: Dense
OutputLayer: Dense

ref: 'https://i.imgur.com/fv1GvFJ.png'

```
1. all are Conv1D layers with any number of filter and filter sizes, there is no
restriction on this
```

```
   restriction on this.

   2. use concatenate layer is to concatenate all the filters/channels.

   3. You can use any pool size and stride for maxpooling layer.

   4. Don't use more than 16 filters in one Conv layer becuase it will increase the no of
   params.
   ( Only recommendation if you have less computing power )

   5. You can use any number of layers after the Flatten Layer.
```

embedding:

In [ ]:

```
!wget --header="Host: media.githubusercontent.com" --header="User-Agent: Mozilla/5.0 (Windows NT 1
0.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.135 Safari/537.36" --head
er="Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/s
d-exchange;v=b3;q=0.9" --header="Accept-Language: en-IN,en-GB;q=0.9,en-US;q=0.8,en;q=0.7" --header
="Referer: https://github.com/allenai/spv2/blob/master/model/glove.6B.100d.txt.gz"
"https://media.githubusercontent.com/media/allenai/spv2/master/model/glove.6B.100d.txt.gz" -c -O '
glove.6B.100d.txt.gz'
```

```
--2020-08-28 01:34:26--
https://media.githubusercontent.com/media/allenai/spv2/master/model/glove.6B.100d.txt.gz
Resolving media.githubusercontent.com (media.githubusercontent.com)... 151.101.0.133,
151.101.64.133, 151.101.128.133, ...
Connecting to media.githubusercontent.com (media.githubusercontent.com)|151.101.0.133|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 134409071 (128M) [application/octet-stream]
Saving to: ‘glove.6B.100d.txt.gz’

glove.6B.100d.txt.g 100%[===================>] 128.18M   217MB/s    in 0.6s

2020-08-28 01:34:32 (217 MB/s) - ‘glove.6B.100d.txt.gz’ saved [134409071/134409071]

time: 6.05 s
```

In [ ]:

```
! gunzip glove.6B.100d.txt.gz
```

```
time: 3.05 s
```

In [ ]:

```
embeddings_index = {}
f = open(os.path.join('glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))
```

```
Found 400000 word vectors.
time: 9.39 s
```

In [ ]:

```
word_index=token_.word_index
embedding_matrix = np.zeros((len(word_index) + 1, 100))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```

```
time: 131 ms
```

embedding layer:

In [ ]:

```
from keras.layers import Embedding

embedding_layer = Embedding(len(word_index) + 1,
                            100,
                            weights=[embedding_matrix],
                            input_length=max_length,
                            trainable=False)
```

```
time: 22.3 ms
```

# Model-1

In [ ]:

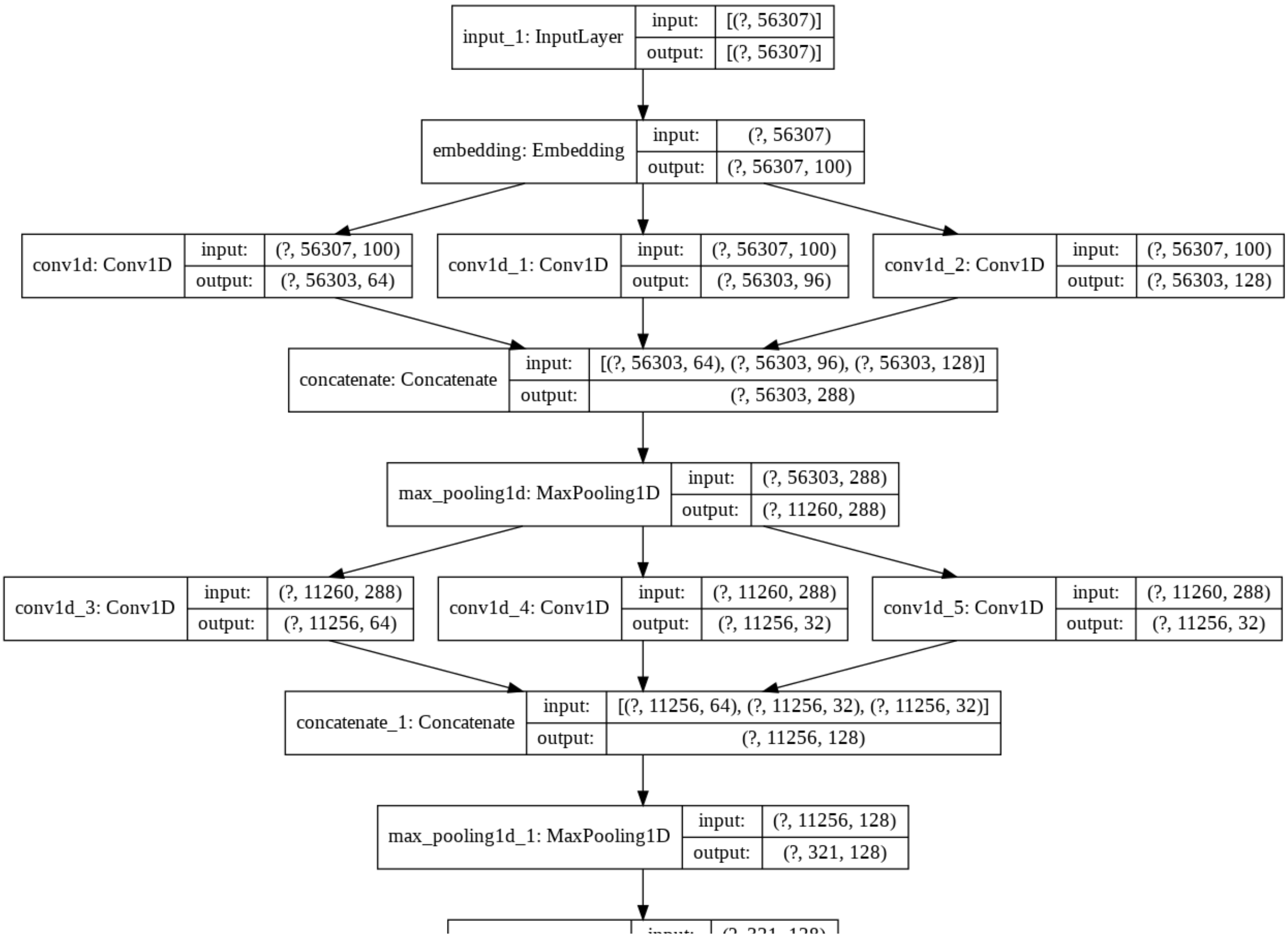
```
#input_length=max_length

#embedding_layer = Embedding(max_length, 300, input_length=max_length, trainable=False)

Input_Layer            =    Input(shape=(max_length,), dtype='int32')
Embedded_Layer         =    embedding_layer(Input_Layer)
Conv_Layer_1           =    Conv1D(64,5, activation='relu',kernel_regularizer=tf.keras.regularizers.
l2(0.01))(Embedded_Layer)
Conv_Layer_2           =    Conv1D(96,5, activation='relu')(Embedded_Layer)
Conv_Layer_3           =    Conv1D(128,5, activation='relu',kernel_regularizer=tf.keras.regularizers
.l2(0.01))(Embedded_Layer)
Concatenate_Layer_1    =    Concatenate()([Conv_Layer_1, Conv_Layer_2,Conv_Layer_3])
Maxpool_layer_1        =    MaxPooling1D(5)(Concatenate_Layer_1)
Conv_Layer_4           =    Conv1D(64,5, activation='relu')(Maxpool_layer_1)
Conv_Layer_5           =    Conv1D(32,5, activation='relu')(Maxpool_layer_1)
Conv_Layer_6           =    Conv1D(32,5, activation='relu')(Maxpool_layer_1)
Concatenate_Layer_2    =    Concatenate()([Conv_Layer_4, Conv_Layer_5,Conv_Layer_6])
Maxpool_layer_2        =    MaxPooling1D(35)(Concatenate_Layer_2)
Conv_Layer_6           =    Conv1D(64,5, activation='relu')(Maxpool_layer_2)
flat_layer             =    Flatten()(Conv_Layer_6)
Drop_out_layer_1       =    Dropout(0.5)(flat_layer)
Dense_layer_1          =    Dense(32,activation='relu')(Drop_out_layer_1)
Output_Layer           =    Dense(20,activation="softmax")(Dense_layer_1)
model = Model(inputs=Input_Layer,outputs=Output_Layer)


model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=0.001),
              metrics=['acc'])
model.summary()
```

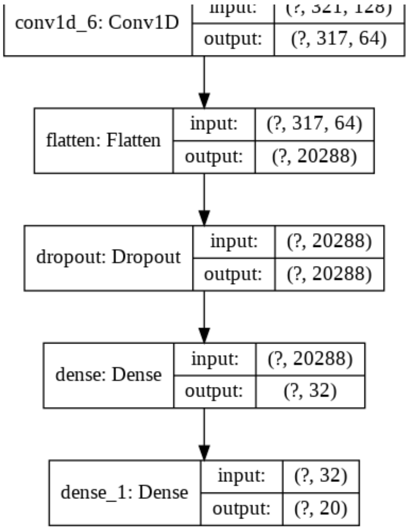
```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 56307)]      0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 56307, 100)   8801900     input_1[0][0]
__________________________________________________________________________________________________
conv1d (Conv1D)                 (None, 56303, 64)    32064       embedding[4][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 56303, 96)    48096       embedding[4][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 56303, 128)   64128       embedding[4][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 56303, 288)   0           conv1d[0][0]
                                                                 conv1d_1[0][0]
                                                                 conv1d_1[0][0]
                                                                 conv1d_2[0][0]
__________________________________________________________________________________________________
max_pooling1d (MaxPooling1D)    (None, 11260, 288)   0           concatenate[0][0]
__________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 11256, 64)    92224       max_pooling1d[0][0]
__________________________________________________________________________________________________
conv1d_4 (Conv1D)               (None, 11256, 32)    46112       max_pooling1d[0][0]
__________________________________________________________________________________________________
conv1d_5 (Conv1D)               (None, 11256, 32)    46112       max_pooling1d[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 11256, 128)   0           conv1d_3[0][0]
                                                                 conv1d_4[0][0]
                                                                 conv1d_5[0][0]
__________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 321, 128)     0           concatenate_1[0][0]
__________________________________________________________________________________________________
conv1d_6 (Conv1D)               (None, 317, 64)      41024       max_pooling1d_1[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 20288)        0           conv1d_6[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 20288)        0           flatten[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 32)           649248      dropout[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 20)           660         dense[0][0]
==================================================================================================
Total params: 9,821,568
Trainable params: 1,019,668
Non-trainable params: 8,801,900
__________________________________________________________________________________________________
time: 148 ms
```

In [ ]:

```
tf.keras.utils.plot_model(
    model, to_file='model.png', show_shapes=True
)
```

Out[ ]:





```
time: 521 ms
```

In [ ]:

```
%reload_ext tensorboard
```

```
time: 2.23 ms
```

In [ ]:

```
!mkdir model__1
```

```
time: 183 ms
```

In [ ]:

```
from keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler
from keras.callbacks import Callback
from keras.callbacks import TensorBoard
from sklearn.metrics import f1_score

import datetime

class Custom_callback(keras.callbacks.Callback):
    def __init__(self, validation_data=()):
        super(keras.callbacks.Callback, self).__init__()

        self.X_val, self.y_val = validation_data

    def on_train_begin(self, logs={}):
        self.microf1score = []
    def on_epoch_end(self, validation_data=(), logs={}):
        y_target = self.y_val
        y_predict = (np.asarray(self.model.predict(self.X_val))).round()
        F1_Score = f1_score(y_target, y_predict,average='micro')
        self.microf1score.append(F1_Score)
        print (" - F1_Score: {0} ".format(F1_Score))
        acc = logs.get('val_acc')
        ## As excepted accuray is 70%
        if ((acc*100) > 70.0) :
          print("\nReached {0} accuracy, so stopping training!!".format((acc*100)))
          self.model.stop_training = True


tensorboard_callback = TensorBoard(log_dir='model__1',histogram_freq=1, write_graph=True,write_grad
s=True)

# This callback will stop the training when there is no improvement in
# the validation loss for three consecutive epochs.

filepath = "best_model_1.h5"
model_chkpt = ModelCheckpoint(filepath, monitor = "val_acc", save_best_only=True, verbose = 1)
```

```
F1 = Custom_callback(validation_data=(X_test, y_test))


callback=[F1,tensorboard_callback,model_chkpt]
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
time: 23 ms
```

In [ ]:

```
import keras
keras.backend.clear_session()
```

```
time: 5.38 ms
```

In [ ]:

```
model.fit(X_train, y_train, validation_data=(X_test, y_test),
          epochs=50, batch_size=32,
          callbacks=callback)
```

```
Epoch 1/50
442/442 [==============================] - ETA: 0s - loss: 2.6571 - acc: 0.1432 — F1_Score: 0.1222
4753227031131

Epoch 00001: val_acc improved from -inf to 0.24857, saving model to best_model_1.h5
442/442 [==============================] - 536s 1s/step - loss: 2.6571 - acc: 0.1432 - val_loss: 2
.0436 - val_acc: 0.2486
Epoch 2/50
442/442 [==============================] - ETA: 0s - loss: 1.8657 - acc: 0.3226 — F1_Score: 0.3170
14057198255

Epoch 00002: val_acc improved from 0.24857 to 0.42023, saving model to best_model_1.h5
442/442 [==============================] - 534s 1s/step - loss: 1.8657 - acc: 0.3226 - val_loss: 1
.5920 - val_acc: 0.4202
Epoch 3/50
442/442 [==============================] - ETA: 0s - loss: 1.4422 - acc: 0.4664 — F1_Score: 0.4849
722789611905

Epoch 00003: val_acc improved from 0.42023 to 0.54408, saving model to best_model_1.h5
442/442 [==============================] - 533s 1s/step - loss: 1.4422 - acc: 0.4664 - val_loss: 1
.2889 - val_acc: 0.5441
Epoch 4/50
442/442 [==============================] - ETA: 0s - loss: 1.1654 - acc: 0.5670 — F1_Score: 0.5086
37779665817

Epoch 00004: val_acc improved from 0.54408 to 0.56597, saving model to best_model_1.h5
442/442 [==============================] - 534s 1s/step - loss: 1.1654 - acc: 0.5670 - val_loss: 1
.2026 - val_acc: 0.5660
Epoch 5/50
442/442 [==============================] - ETA: 0s - loss: 1.0002 - acc: 0.6280 — F1_Score: 0.5813
232717962539

Epoch 00005: val_acc improved from 0.56597 to 0.61610, saving model to best_model_1.h5
442/442 [==============================] - 534s 1s/step - loss: 1.0002 - acc: 0.6280 - val_loss: 1
.1745 - val_acc: 0.6161
Epoch 6/50
442/442 [==============================] - ETA: 0s - loss: 0.8634 - acc: 0.6793 — F1_Score: 0.6421
118793211815

Epoch 00006: val_acc improved from 0.61610 to 0.65902, saving model to best_model_1.h5
442/442 [==============================] - 533s 1s/step - loss: 0.8634 - acc: 0.6793 - val_loss: 1
.0638 - val_acc: 0.6590
Epoch 7/50
442/442 [==============================] - ETA: 0s - loss: 0.7329 - acc: 0.7296 — F1_Score: 0.6884
144878867834

Epoch 00007: val_acc improved from 0.65902 to 0.69365, saving model to best_model_1.h5
442/442 [==============================] - 533s 1s/step - loss: 0.7329 - acc: 0.7296 - val_loss: 0
.9900 - val_acc: 0.6936
```

```
Epoch 8/50
442/442 [==============================] - ETA: 0s - loss: 0.6416 - acc: 0.7558 — F1_Score: 0.6780
674479789525

Epoch 00008: val_acc did not improve from 0.69365
442/442 [==============================] - 533s 1s/step - loss: 0.6416 - acc: 0.7558 - val_loss: 1
.0408 - val_acc: 0.6779
Epoch 9/50
442/442 [==============================] - ETA: 0s - loss: 0.5665 - acc: 0.7872 — F1_Score: 0.7045
375822653274

Reached 70.21457552909851 accuracy, so stopping training!!

Epoch 00009: val_acc improved from 0.69365 to 0.70215, saving model to best_model_1.h5
442/442 [==============================] - 533s 1s/step - loss: 0.5665 - acc: 0.7872 - val_loss: 1
.0401 - val_acc: 0.7021
```

Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7ff36c359c50>
```

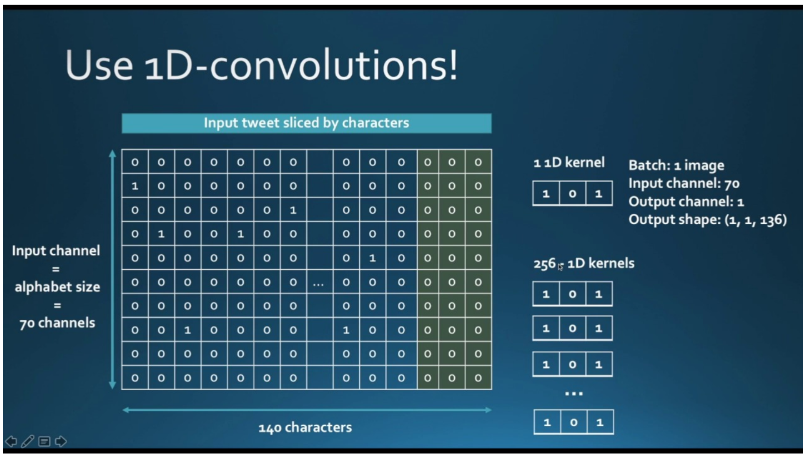
```
time: 1h 20min 11s
```

In [ ]:

```
%tensorboard --logdir 'model__1'
```

```
time: 3.6 s
```

## Model-2 : Using 1D convolutions with character embedding



Use 1D-convolutions!

Input tweet sliced by characters

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |  | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |  | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | 0 | 1 | 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 | 1 | 0 | 0 | 0 | 0 |  | 1 | 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 |

Input channel<br>
=<br>
alphabet size<br>
=<br>
70 channels

140 characters

1 1D kernel

| 1 | 0 | 1 |
|---|---|---|

Batch: 1 image
Input channel: 70
Output channel: 1
Output shape: (1, 1, 136)

256 1D kernels

| 1 | 0 | 1 |
|---|---|---|
| 1 | 0 | 1 |
| 1 | 0 | 1 |
| ... |  |  |
| 1 | 0 | 1 |

Here are the some papers based on Char-CNN

1. Xiang Zhang, Junbo Zhao, Yann LeCun. Character-level Convolutional Networks for Text Classification.NIPS 2015
2. Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush. Character-Aware Neural Language Models. AAAI 2016
3. Shaojie Bai, J. Zico Kolter, Vladlen Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling
4. Use the pratrained char embeddings https://github.com/minimaxir/char-embeddings/blob/master/glove.840B.300d-char.txt

In [ ]:

```
keras.backend.clear_session()
```

In [ ]:

In [ ]:

```
!wget --header="Host: codeload.github.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.135 Safari/537.36" --header="A
ccept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/s
d-exchange;v=b3;q=0.9" --header="Accept-Language: en-IN,en-GB;q=0.9,en-US;q=0.8,en;q=0.7" --header
="Referer: https://github.com/minimaxir/char-embeddings" --header="Cookie:
_octo=GH1.1.822105914.1592881801; _ga=GA1.2.1585639001.1592881801; logged_in=yes;
dotcom_user=MOPARTHISATISH69; tz=Asia%2FCalcutta; _gat=1" --header="Connection: keep-alive"
"https://codeload.github.com/minimaxir/char-embeddings/zip/master" -c -O 'char-embeddings-
master.zip'
```

```
--2020-08-28 05:59:31--  https://codeload.github.com/minimaxir/char-embeddings/zip/master
Resolving codeload.github.com (codeload.github.com)... 140.82.121.10
Connecting to codeload.github.com (codeload.github.com)|140.82.121.10|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: ‘char-embeddings-master.zip’

char-embeddings-mas     [    <=>             ]  10.90M  7.76MB/s    in 1.4s

2020-08-28 05:59:32 (7.76 MB/s) - ‘char-embeddings-master.zip’ saved [11431744]

time: 1.8 s
```

In [ ]:

```
# Now Split the data into Train and test. use 25% for test also do a stratify split.

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data["data"], data["text_class"],stratify=data[
"text_class"],test_size=0.25)
```

```
time: 42.8 ms
```

In [ ]:

```
t  = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}~\t\n',char_level = True) #will filter
everything except underscore'_'
t.fit_on_texts(X_train)
X_train_tokenised = t.texts_to_sequences(X_train)
X_test_tokenised = t.texts_to_sequences(X_test)


print(len(X_train_tokenised[0]))
print(len(X_test_tokenised))
```

```
277
4707
time: 6.84 s
```

In [ ]:

```
max_length=0
for i in X_train_tokenised:
  if (len(i)>max_length):
    max_length=len(i)

print("max length:",max_length)
```

```
max length: 50069
time: 4.48 ms
```

In [ ]:

```
Max_len = data['data'].astype('str')
len_list = []
for each_text in Max_len:
  split = each_text.split()
  len_list.append(len(split))
  len_list.append(len(split))
print("The length of the list", len(len_list))

print('90th to 100 percentile')
for i in range(90,101,1):
  print(i,'percentile value is', np.percentile(len_list, i))
print("="*50)
print('99 to 100th percentile')
for i in range(10,110,10):
  print(99+(i/100),'percentile value is',np.percentile(len_list,99+(i/100)))
```

```
The length of the list 18828
90th to 100 percentile
90 percentile value is 353.2999999999993
91 percentile value is 377.0
92 percentile value is 403.0
93 percentile value is 439.0
94 percentile value is 479.0
95 percentile value is 533.0
96 percentile value is 601.9199999999983
97 percentile value is 723.1899999999987
98 percentile value is 949.4599999999991
99 percentile value is 1464.4599999999991
100 percentile value is 8859.0
==================================================
99 to 100th percentile
99.1 percentile value is 1555.140000000014
99.2 percentile value is 1721.2239999999802
99.3 percentile value is 1808.531999999992
99.4 percentile value is 2119.418000000005
99.5 percentile value is 2366.705000000027
99.6 percentile value is 2917.8959999999206
99.7 percentile value is 3403.076000000001
99.8 percentile value is 4532.346000000001
99.9 percentile value is 6101.186000000205
100.0 percentile value is 8859.0
time: 231 ms
```

In [ ]:

```
max_length = 18000
```

```
time: 875 µs
```

In [ ]:

```
Xtrain = pad_sequences(X_train_tokenised, maxlen=max_length, padding='post')

Xtest = pad_sequences(X_test_tokenised, maxlen=max_length, padding='post')
```

```
time: 3.09 s
```

In [ ]:

```
embedding_vectors = {}
with open(os.path.join('glove.840B.300d-char.txt'), 'r') as f:
    for line in f:
        line_split = line.strip().split(" ")
        vector = np.array(line_split[1:], dtype=float)
        char = line_split[0]
        embedding_vectors[char] = vector
```

In [ ]:

```
embedding_vectors = {}
with open(os.path.join('glove.6B.100d.txt'), 'r') as f:
    for line in f:
        line_split = line.strip().split(" ")
        vec = np.array(line_split[1:], dtype=float)
        char = line_split[0]
        embedding_vectors[char] = vec
```

```
time: 9.69 s
```

In [ ]:

```
word_index=t.word_index
print(len(word_index))
```

```
39
time: 886 µs
```

embedding layer 2:

In [ ]:

```
embedding_matrix = np.zeros((len(word_index) + 1, 100))
for char, i in word_index.items():
    #print ("{}, {}".format(char, i))
    embedding_vector = embedding_vectors.get(char)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
time: 4.42 ms
```

In [ ]:

```
from keras.layers import Embedding

embedding_layer2 = Embedding(len(word_index) + 1,
                            100,
                            weights=[embedding_matrix],
                            input_length=max_length,
                            trainable=False)
```

```
time: 2.4 ms
```

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense,Dropout,Input
from keras.layers import Flatten,BatchNormalization
from keras.layers import Embedding
from keras.layers import Concatenate
from keras.models import Model
from keras.optimizers import Adam
from keras import regularizers


from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
```

```
time: 2.68 ms
```

In [ ]:

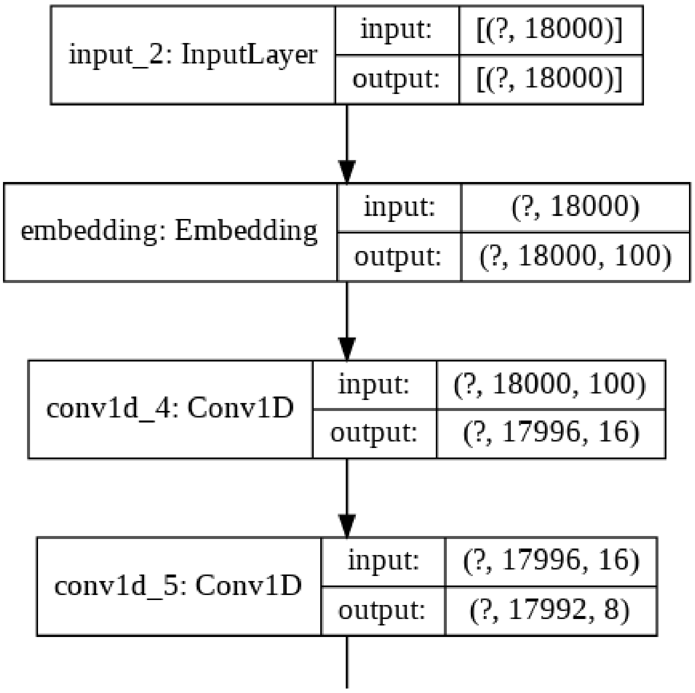
```
max_length = 18000

Input_Layer                        =  Input(shape=(max_length,), dtype='int32')
Embedded_Layer                     =  embedding_layer2(Input_Layer)
Convolution_layer_1                =  Conv1D(16,5, activation='relu')(Embedded_Layer)
Convolution_layer_2                =  Conv1D(8,5, activation='relu')(Convolution_layer_1)
Maxpool_layer_1                    =  MaxPooling1D(5)(Convolution_layer_2)
Convolution_layer_3                =  Conv1D(4,5, activation='relu')(Maxpool_layer_1)
Convolution_layer_4                =  Conv1D(8,5, activation='relu')(Convolution_layer_3)
Maxpool_layer_2                    =  MaxPooling1D(35)(Convolution_layer_4)
Flat_Layer                         =  Flatten()(Maxpool_layer_2)
Dropout_layer                      =  Dropout(0.2)(Flat_Layer)
Dense_layer                        =  Dense(128,activation='relu')(Dropout_layer)
Output_layer                       =  Dense(20,activation="softmax")(Dense_layer)
Output_layer                          =   Dense(20,activation="softmax")(Dense_layer)
model                                 =   Model(inputs=Input_Layer,outputs=Output_layer)

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=0.001),
              metrics=['acc'])
model.summary()
```

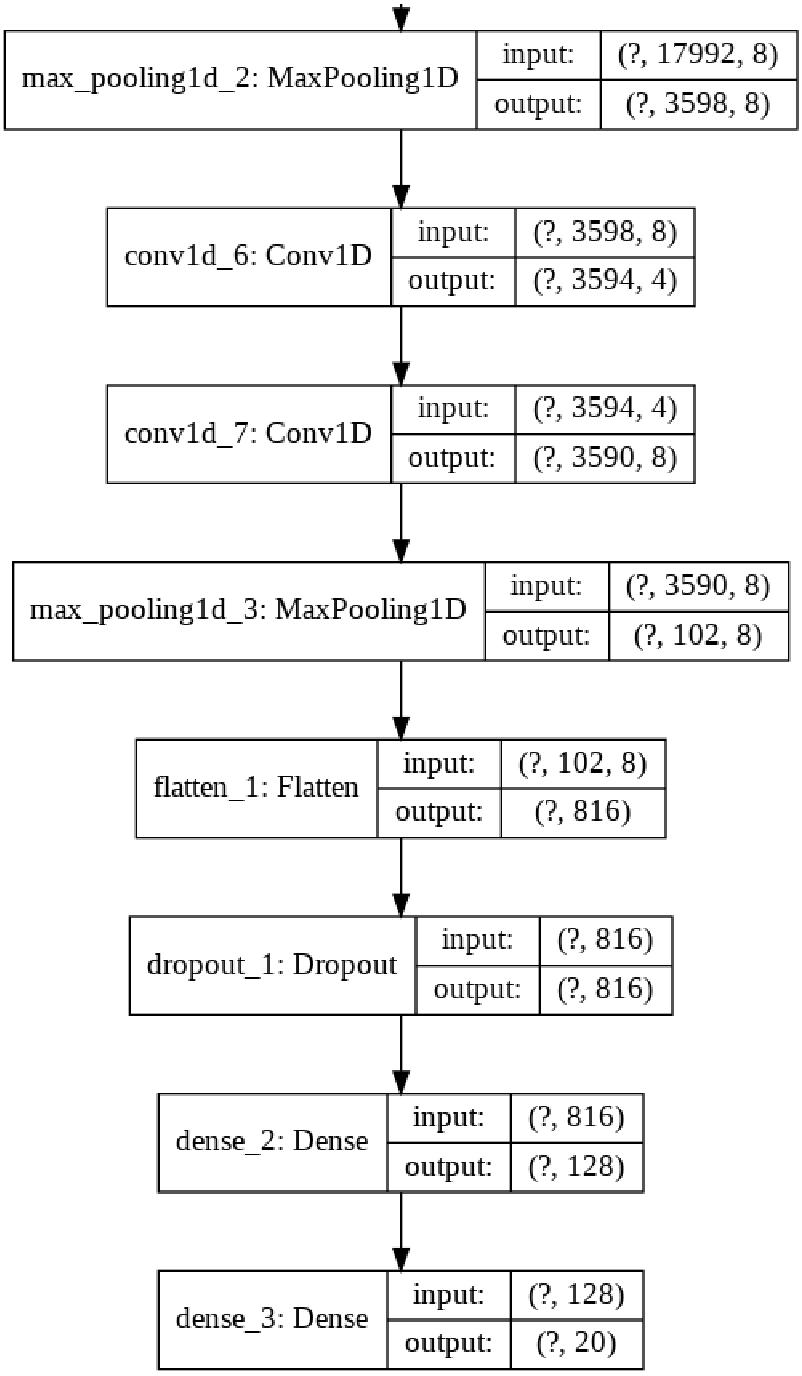
```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, 18000)]           0         
_________________________________________________________________
embedding (Embedding)        (None, 18000, 100)        4000      
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 17996, 16)         8016      
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 17992, 8)          648       
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 3598, 8)           0         
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 3594, 4)           164       
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 3590, 8)           168       
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 102, 8)            0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 816)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 816)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 128)               104576    
_________________________________________________________________
dense_3 (Dense)              (None, 20)                2580      
=================================================================
Total params: 120,152
Trainable params: 116,152
Non-trainable params: 4,000
_________________________________________________________________
time: 103 ms
```

In [ ]:

```
tf.keras.utils.plot_model(
    model, to_file='model.png', show_shapes=True
)
```

Out[ ]:





```
time: 479 ms
```

In [ ]:

```
!mkdir model_2
```

```
time: 223 ms
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder

def prepare_targets(y_train, y_test):
 le = LabelEncoder()
 le.fit(y_train)
 y_train = le.transform(y_train)
 y_test = le.transform(y_test)
 return y_train, y_test

y_train, y_test=prepare_targets(y_train, y_test)
print(np.unique(y_train))
print(y_train[0:10])
```

```
ytrain_cat = keras.utils.to_categorical(y_train, num_classes=20)
ytest_cat = keras.utils.to_categorical(y_test, num_classes=20)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
[ 5 16 13 11  3  9 12 11 17 11]
time: 17.2 ms
```

In [ ]:

```
from keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler
from keras.callbacks import Callback
from keras.callbacks import TensorBoard
from sklearn.metrics import f1_score

import datetime

class Custom_callback(keras.callbacks.Callback):

    def __init__(self,validation_data):
      super(Custom_callback,self).__init__()
      self.x_test = validation_data[0]
      self.y_test = validation_data[1]

    def on_train_begin(self, logs={}):
        self.f1Score_List = []
    def on_epoch_end(self, epoch, logs={}):
        #print("system lr :",self.model.optimizer.lr)
        y_predict = (np.asarray(self.model.predict(self.x_test))).round()
        y_targ = self.y_test
        f1Score = f1_score(y_targ, y_predict,average='micro')
        self.f1Score_List.append(f1Score)
        print (" — F1_Score: %f "%(f1Score))
        acc = logs.get('val_acc')
        if ((acc*100) > 10.3) :
          print("\nReached {0} accuracy, so stopping training!!".format((acc*100)))
          self.model.stop_training = True


tensorboard_callback = TensorBoard(log_dir='model_2',histogram_freq=1, write_graph=True,write_grads
=True)

filepath = "best_model_2.h5"
model_chkpt = ModelCheckpoint(filepath, monitor = "val_acc", save_best_only=True, verbose = 1)

validation_data=(Xtest, ytest_cat)

callback=[Custom_callback(validation_data),tensorboard_callback,model_chkpt]
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
time: 21.1 ms
```

In [ ]:

```
model.fit(Xtrain, ytrain_cat, validation_data=(Xtest, ytest_cat),
          epochs=30, batch_size=8,
          callbacks=callback)
```

```
Epoch 1/30
   2/1766 [..............................] - ETA: 1:53 - loss: 3.0301 - acc:
0.0625WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch time
(batch time: 0.0215s vs `on_train_batch_end` time: 0.1064s). Check your callbacks.
1766/1766 [==============================] - ETA: 0s - loss: 2.9404 - acc: 0.0783 — F1_Score: 0.00
0000

Epoch 00001: val_acc improved from -inf to 0.08562, saving model to best_model_2.h5
1766/1766 [==============================] - 49s 28ms/step - loss: 2.9404 - acc: 0.0783 - val_loss
: 2.9302 - val_acc: 0.0856
Epoch 2/30
1765/1766 [============================>.] - ETA: 0s - loss: 2.8758 - acc: 0.0984 — F1_Score: 0.05
4343
```

```
Reached 10.919906198978424 accuracy, so stopping training!!

Epoch 00002: val_acc improved from 0.08562 to 0.10920, saving model to best_model_2.h5
1766/1766 [==============================] - 48s 27ms/step - loss: 2.8758 - acc: 0.0984 - val_loss
: 2.8372 - val_acc: 0.1092
```

```
Out[ ]:
```

```
<tensorflow.python.keras.callbacks.History at 0x7fef281ef320>
```

```
time: 1min 38s
```

```
In [ ]:
```

```
%tensorboard --logdir 'model_2'
```

```
time: 9.64 s
```

**Observations**

# Model-1

1. For Model - 1 I used maxlen = 56307, as it is the highest length of essay in data.
2. I used “relu” activation function for all layers (Cov1D and Dense) and added L2 regularizes and dropout to avoid overfit.
3. Wrote custom callbacks to stop the training once the validation accuracy reaches to 70+% as it is fluctuating and sometimes seems to be overfit.
4. By observing the values, it seems to be slight overfitting for both of the accuracy’s and steady increase in F1-score.

# Model-2

I used maxlen = 18000, because with the max length of 56307 Google Colab is getting crashed, I dropped a mail to team for suggestions. Done some EDA and found the 99% value - 6101 and tried with it but Got dimension mismatch error while training. So used 18000, (~1/3*56307) it worked well.

I used “relu” activation function for all layers (Cov1D and Dense). Used only dropout with 20%toavoid overfit. Our model gave 10+%validation accuracy in 2 epochs. Wrote custom callbacks to stop the training once the validation accuracy reaches to 10+%. Model-2 is neither overfit nor underfit.

**Reference : -**

1. reference pdf
2. Slack conversations