Research of film script generation based on the text of fiction book

Author: Danylo Iaroslavovych Ivanchyshyn (student of the SA-32 group)

Corporate email: danylo.ivanchyshyn.sa.2018@lpnu.ua

Supervisor: Ph.D., Associate Professor Victoriia Anatoliivna Vysotska

Name and address of the organization: Lviv Polytechnic National University, 79013, L'viv, Stepana Banderi, 12 str.

Content:

Introduction

In this paper, we will try to develop a system that will help companies in creating movies based on art books.

For five years (until 2021), consumer interest in going to the cinema among many developed countries remained unchanged in the global film production and distribution industry, following the pattern of stagnation observed up to that period. In this regard, more and more companies are looking for simpler and more reliable ways of film production.

In this paper, we will try to get a little closer to this goal, namely to develop a system that will help companies in creating movies based on art books

Goal

The purpose of this course work is to develop a concept of the system that will automate the generation of screenplays based on the art book.

Task

To solve this problem you need to solve the following subtasks:

1. Evaluate the available methods of solving the problem;
2. Analyze the advantages and disadvantages of existing solutions;
3. Build all system components and synchronize their work;

4. Test the system for efficiency.

The object of research is the generation of a screenplay based on a book.

The subject of research is the process of generating a screenplay based on a book.

The main goal of our design is to create a product that will provide the greatest productivity, ease of use and speed when writing a screenplay.

Practical significance of the obtained results: after the development of this system, it can be improved for further use in different companies.

Relevance and novelty:

the use of AI in the film industry has increased significantly. Deep Learning makes its mark everywhere from the healthcare sector to the entertainment sector, for some time now Deep Leaning has been releasing news to predict box office failure or movie success or to write a whole movie script using services like Cinelytic and ScriptBook as well. offer in-depth learning tools for scenario and scenario analysis. We submit the data to a machine learning model to identify specific trends and patterns, and then transfer everything back to the script.

So, let's start our study.

First, download the necessary libraries:

In [1]:

```
import  numpy  as  np
import  pandas  as  pd
import  string
import  spacy

from  matplotlib.pyplot  import  imread
from  matplotlib  import  pyplot  as  plt
from  wordcloud  import  WordCloud
% matplotlib inline
```

I. Let's start our analysis with a preview of the data:

Return to table of contents

In [2]:

```
import  os
for  dirname ,  _ ,  filenames  in  os . walk ( 'D: \\ backup \\ CA-32 \\ Course (ma
    for  filename  in  filenames :
        print ( os . path . join ( dirname ,  filename ))
```

```
D: \ backup \ CA-32 \ Course (machine learning) \ 1.png
D: \ backup \ CA-32 \ Coursework (machine learning) \ Coursework_automated_screenpla
y_generation_presentation1.pptx
D: \ backup \ CA-32 \ Coursework (machine learning) \ Coursework_automated_screenpla
y_generation_presentation2.pptx
D: \ backup \ CA-32 \ Course (machine learning) \ data.csv
D: \ backup \ CA-32 \ Course (machine learning) \ data.tsv
D: \ backup \ CA-32 \ Course (machine learning) \ data.xlsx
D: \ backup \ CA-32 \ Course (machine learning) \ data1.tsv
D: \ backup \ CA-32 \ Course (machine learning) \ data2.tsv
```

```
D: \ backup \ CA-32 \ Course (machine learning) \ data3.tsv
D: \ backup \ CA-32 \ Course (machine learning) \ don.jpg
D: \ backup \ CA-32 \ Course (machine learning) \ don1.png
D: \ backup \ CA-32 \ Course (machine learning) \ Puzo Mario-The Godfather-Script.tx
t
D: \ backup \ CA-32 \ Course (machine learning) \ Puzo Mario-The Godfather.txt
D: \ backup \ CA-32 \ Course (machine learning) \ scenario_generation_on_book_ipynb.
html
D: \ backup \ CA-32 \ Course (machine learning) \ The Godfather.html
D: \ backup \ CA-32 \ Course (machine learning) \ Untitled.png
D: \ backup \ CA-32 \ Course (machine learning) \ Курсова_робота_100% _Іванчишин_CA-
32 (ML) .pdf
D: \ backup \ CA-32 \ Course (machine learning) \ Курсова_робота_55% _Іванчишин_CA-3
2 (ML) .pdf
D: \ backup \ CA-32 \ Course (machine learning) \ Курсова_робота_88% _Іванчишин_CA-3
2 (ML) .pdf
D: \ backup \ CA-32 \ Course (machine learning) \ Курсова_робота_Іванчишин_CA-32 (M
L) .docx
```

In [3]:

```
filename = 'D: \\ backup \\ CA-32 \\ Course (machine learning) \\ Puzo Mario-The G
with open ( filename , "r" , encoding = "UTF-8" ) as f :
    book = f . readlines ()
```

In [4]:

```
len ( book )
```

Out [4]:

```
5821
```

Let's look at the first ten characters that can be thrown away because they do not affect the content.

In [5]:

```
book [ 0 : 12 ]
```

Out [5]:

```
['The Godfather \ n',
 '\ n',
 'by Mario Puzo \ n',
 '\ n',
 'For Anthony Cleri \ n',
 '\ n',
 '\ n',
 '\ n',
 '\ n',
 'Book One \ n',
 '\ n',
 '\ n']
```

II. Data cleaning

Return to table of contents

We must always clear the data before applying machine learning or a statistical model. No model gives significant results with chaotic data. it is the process of detecting and correcting (or deleting) damaged or inaccurate records from a set of records, tables or databases and involves identifying incomplete, incorrect, inaccurate or irrelevant pieces of data and then replacing, modifying or deleting dirty or crude data.

Fortunately, a book is usually already a blank document that reviews spelling, editing, grammar, and so on. Thus, the words and sentences we receive are usually read without errors. Unlike the answers in some questionnaires, where people can write anything and make a lot of mistakes, even if they don't want to. But still there are some extra parts of the text that we do not need in our analysis, so let's move on to them.

1. Empty lines

Return to table of contents

We do not need blank lines or a blank line in our text because they do not contain any information. Therefore, it is easiest to delete them at the beginning.

In [6]:

```
# remove line spacing

book = [ x . strip () for x in book ]

# delete empty characters because they are considered False by Python

book = [ x for x in book if x ]
book [ 4 : 10 ]
```

Out [6]:

```
['Chapter 1',
 'Behind every great fortune there is a crime.',
 'Balzac',
 'Amerigo Bonasera sat in New York Criminal Court Number 3 and waited for justice; v
engeance on the men who had so cruelly hurt his daughter, who had tried to dishonor
 her. ',
 'The judge, a formidably heavy-featured man, rolled up the sleeves of his black rob
e as if to physically chastise the two young men standing before the bench. His face
was cold with majestic contempt. But there was something false in all this that Amer
igo Bonasera sensed but did not yet understand. ',
 '"You acted like the worst kind of degenerates," the judge said harshly. Yes, yes,
 thought Amerigo Bonasera. Animals. Animals. The two young men, glossy hair crew cu
t, scrubbed clean-cut faces composed into humble contrition, bowed their heads in su
bmission. ']
```

2. Remove unnecessary pieces of text from the book

Return to table of contents

We need to analyze the text of the book, not the author, the title of the book or the year of publication. Therefore, we will remove the extra parts from the text for analysis.

In [7]:

```
core_book = book [ 7 :]
core_book [ 0 : 8 ]
```

Out [7]:

```
['Amerigo Bonasera sat in New York Criminal Court Number 3 and waited for justice; v
engeance on the men who had so cruelly hurt his daughter, who had tried to dishonor
 her. ',
 'The judge, a formidably heavy-featured man, rolled up the sleeves of his black rob
e as if to physically chastise the two young men standing before the bench. His face
was cold with majestic contempt. But there was something false in all this that Amer
igo Bonasera sensed but did not yet understand. ',
 '"You acted like the worst kind of degenerates," the judge said harshly. Yes, yes,
 thought Amerigo Bonasera. Animals. Animals. The two young men, glossy hair crew cu
t, scrubbed clean-cut faces composed into humble contrition, bowed their heads in su
bmission. ',
 'The judge went on. "You acted like wild beasts in a jungle and you are fortunate y
ou didn't sexually molest that poor girl or I'd put you behind bars for twenty year
s." The judge paused, his eyes beneath impressively thick brows flickered slyly towa
rd the sallow-faced Amerigo Bonasera, then lowered to a stack of probation reports b
```

```
efore him. He frowned and shrugged as if convinced against his own natural desire. H
e spoke again. ',
 '“But because of your youth, your clean records, because of your fine families, and
because the law in its majesty does not seek vengeance, I hereby sentence you to thr
ee years’ confinement to the penitentiary. Sentence to be suspended. ”',
 'Only forty years of professional mourning kept the overwhelming frustration and ha
tred from showing on Amerigo Bonasera's face. His beautiful young daughter was still
in the hospital with her broken jaw wired together; and now these two animales went
 free? It had all been a farce. He watched the happy parents cluster around their da
rling sons. Oh, they were all happy now, they were smiling now. ',
 'The black bile, sourly bitter, rose in Bonasera's throat, overflowed through tight
ly clenched teeth. He used his white linen pocket handkerchief and held it against h
is lips. He was standing so when the two young men strode freely up the aisle, confi
dent and cool-eyed, smiling, not giving him so much as a glance. He let them pass wi
thout saying a word, pressing the fresh linen against his mouth. ',
 'The parents of the animales were coming by now, two men and two women his age but
 more American in their dress. They glanced at him, shamefaced, yet in their eyes wa
s an odd, triumphant defiance. ']
```

Let's combine all the sentences into a single text:

In [8]:
```
text = ''.join(core_book)
len(text)
```

Out [8]:
```
939761
```

In [9]:
```
text[0:400]
```

Out [9]:
```
'Amerigo Bonasera sat in New York Criminal Court Number 3 and waited for justice; ve
ngeance on the men who had so cruelly hurt his daughter, who had tried to dishonor h
er. The judge, a formidably heavy-featured man, rolled up the sleeves of his black r
obe as if to physically chastise the two young men standing before the bench. His fa
ce was cold with majestic contempt. But there was something false '
```

3. Punctuation

Return to table of contents

In fact, punctuation doesn't help in checking words and their meanings, so let's get rid of that.

In [10]:
```
no_punc_text = text.translate(str.maketrans('', '', string.punctuati
no_punc_text[0:550]
```

Out [10…
```
'Amerigo Bonasera sat in New York Criminal Court Number 3 and waited for justice ven
geance on the men who had so cruelly hurt his daughter who had tried to dishonor her
The judge a formidably heavyfeatured man rolled up the sleeves of his black robe as
 if to physically chastise the two young men standing before the bench His face was
 cold with majestic contempt But there was something false in all this that Amerigo
 Bonasera sensed but did not yet understand “You acted like the worst kind of degene
rates” the judge said harshly Yes yes thought Ame '
```

In [11]:
```
len(text) - len(no_punc_text)
```

Out [11…
```
22312
```

4. Stopwords

Return to table of contents

Stop words are a special case of words that work as a filler and usually do not have much meaning. We will remove them later because we compare their appearance in the text with meaningful words. But let's see what stop words are.

In [12]:

```
from nltk.corpus import stopwords

stop_words = set ( stopwords . words ( "english" ))
print ( stop_words )
```

```
{'hasn', 's', 'o', 'you'll', 'under', 'shouldn', 'wouldn', 'she', 'this', 'further',
'should', 'themselves' , 'where', 'below', 'hadn't', 'that'll', 'until', 'again', 'f
or', 'more', 'yourself', 'some', 'over', ' is ',' them ',' he ',' has ',' on ',' the
irs ', "it's",' can ',' its ',' did ', "won't",' ourselves ',' a ',' d ',' couldn
 ',' above ',' aren't ',' how ',' now ',' be ',' into ',' during ',' so ',' t ',' ag
ainst ',' there ',' their ','such', 'your', 'same', 'my', 'had', 'don', 'll', 'are
n', 've', 'him', 'at', 'few', 'other ',' because ',' own ',' doesn ',' after ',' no
 ',' if ',' what ',' and ',' very ',' needn ',' his', 'wasn', 'won', 'you', 'ain',
 'himself', 'too', 'are', 'up', 'i', 'once', 'were', 'mustn', 'does',' whom ',' here
',' wasn't ',' why ',' both ',' mightn ',' about ',' y ',' out ',' than ',' weren
 ',' as ',' it ',' not ',' needn't ',' have ',' these ','while', 're', 'isn', 'betwe
en', 'from', 'having', 'me', 'which', 'by', 'being', 'you've', 'down', 'an', 'or',
 'itself', 'just', 'herself', 'mustn't', 'myself', 'hadn', 'didn', 'to', 'yours', 'a
ll', 'shan', 'hasn't', 'shouldn't', 'in', 'that', 'isn't', 'our', 'who', 'through',
 'they', 'off', 'when', 'haven', 'most', 'will', 'then', 'yourselves',' doing ', "do
n't",' we ', "weren't",' only ',' but ', "doesn't",' am ',"she's", 'the', 'any', "ha
ven't", 'those', "you'd", "shan't", 'hers',' of ', "couldn't",' nor ',' each ',' bef
ore ', "should've",' been ',' ours', 'do', 'with', "didn't", "wouldn't", "you're",
 'was', 'm', 'ma', 'her', "mightn't"}wouldn't "," you're ", 'was', 'm', 'ma', 'he
r'," mightn't "}wouldn't "," you're ", 'was', 'm', 'ma', 'her'," mightn't "}
```

III. Feature engineering

Return to table of contents

Feature engineering is the process of creating new variables for a given data set with the idea of improving the accuracy of model prediction or a better description of the data set.

Features can be:

- numerical (number of words in a sentence)
- categorical (what is this sentence?)
- boolean (Is the sentence longer than 50 characters? True / False)
- ordinal (sentence short, medium or long?)

1. Tokenization

Return to table of contents

Tokenization is essentially breaking down a phrase, sentence, paragraph, or entire text document into smaller units, such as individual words or terms. Each of these smaller units is called a token.

In [13]:

```
from nltk import word_tokenize

text_tokens = word_tokenize ( no_punc_text )
print ( text_tokens [ 0 : 50 ])
```

```
['Amerigo', 'Bonasera', 'sat', 'in', 'New', 'York', 'Criminal', 'Court', 'Number',
 '3', 'and', 'waited', ' for ',' justice ',' vengeance ',' on ',' the ',' men ',' wh
o ',' had ',' so ',' cruelly ',' hurt ',' his', 'daughter' , 'who', 'had', 'tried',
 'to', 'dishonor', 'her', 'The', 'judge', 'a', 'formidably', 'heavyfeatured', 'man',
' rolled ',' up ',' the ',' sleeves', 'of', 'his',' black ',' robe ',' as', 'if', 't
o', 'physically', 'chastise' ]
```

In [14]:

```
len ( text_tokens )
```

Out [14…

```
184314
```

Once we have marked the text with a token, we can remove the stop words from it.

In [15]:

```
my_stop_words  =  stopwords . words ( 'english' )
my_stop_words . append ( 'the' )
no_stop_tokens  =  [ word  for  word  in  text_tokens  if  not  word  in  my_stop_wo
print ( no_stop_tokens [ 0 : 40 ])
```

```
['Amerigo', 'Bonasera', 'sat', 'New', 'York', 'Criminal', 'Court', 'Number', '3', 'w
aited', 'justice', 'vengeance', ' men ',' cruelly ',' hurt ',' daughter ',' tried
 ',' dishonor ',' The ',' judge ',' formidably ',' heavyfeatured ',' man ',' rolled
 ',' sleeves' , 'black', 'robe', 'physically', 'chastise', 'two', 'young', 'men', 's
tanding', 'bench', 'His',' face ',' cold ',' majestic ',' contempt ',' But ']
```

In [16]:

```
len ( no_stop_tokens )
```

Out [16…

```
103892
```

2. Visualization (wordcloud)

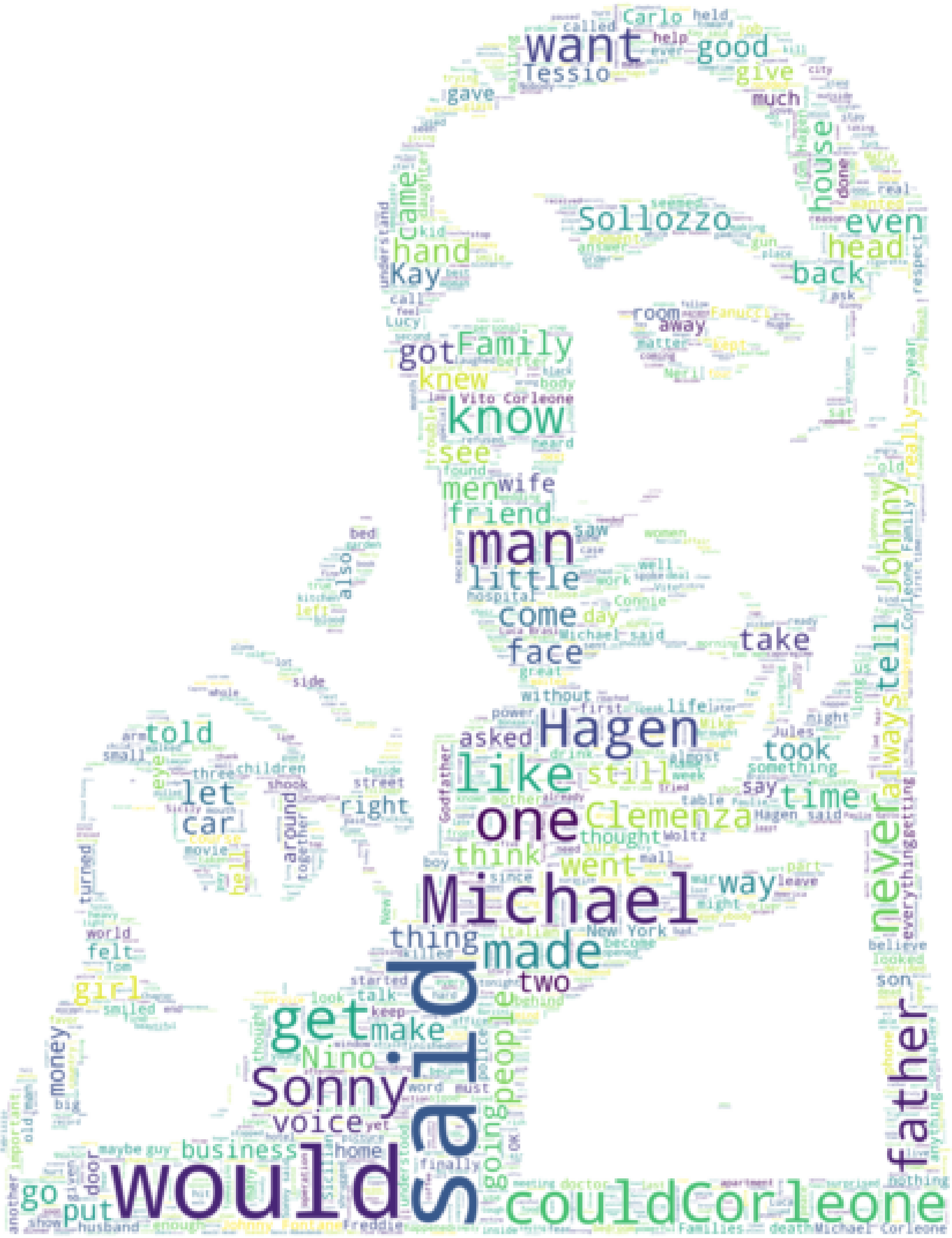
Return to table of contents

Let's make a basic visualization - wordcloud. Further figures will be present in the sub-item on TF-IDF.

In [17]:

```
from  PIL  import  Image
import  numpy  as  np

plt . figure ( figsize = ( 11 ,  10 ))
mask  =  np . array ( Image . open ( 'D: \\ backup \\ CA-32 \\ Course (machine learn
wc  =  WordCloud ( stopwords = stop_words ,
               mask = mask ,  background_color = "white" ,
               max_words = 2000 , max_font_size = 256 ,
               random_state = 42 ,  width = mask . shape [ 1 ],
               height = mask . shape [ 0 ])
wc . generate ( no_punc_text )
plt . imshow ( wc ,  interpolation = "bilinear" )
plt . axis ( 'off' )
plt . show ()
```



2. Lowercase

Return to table of contents

Something as simple as writing lowercase letters in all words is very helpful, because the first letter in a new sentence is uppercase by default, and there are several names of people and things, also with uppercase letters. Localization is standardization.

In [18]:

```
lower_words = [ x . lower () for x in no_stop_tokens ]
print ( lower_words [ 0 : 25 ])
```

```
['amerigo', 'bonasera', 'sat', 'new', 'york', 'criminal', 'court', 'number', '3', 'w
aited', 'justice', 'vengeance', ' men ',' cruelly ',' hurt ',' daughter ',' tried
```

```
 ',' dishonor ',' the ',' judge ',' formidably ',' heavyfeatured ',' man ',' rolled
 ',' sleeves' ]
```

In linguistic morphology and information retrieval, stemming is the process of reducing conjugated (or sometimes derived) words to their base, base, or root words — usually the written word.

In [19]:

```
from nltk.stem import PorterStemmer

ps = PorterStemmer ()
stemmed_tokens = [ ps . stem ( word ) for word in lower_words ]
print ( stemmed_tokens [ 0 : 40 ])
```

```
['amerigo', 'bonasera', 'sat', 'new', 'york', 'crimin', 'court', 'number', '3', 'wai
t', 'justic', 'vengeanc', ' men ',' cruelli ',' hurt ',' daughter ',' tri ',' dishon
or ',' the ',' judg ',' formid ',' heavyfeatur ',' man ',' roll ',' sleev ' , 'blac
k', 'robe', 'physic', 'chastis',' two ',' young ',' men ',' stand ',' bench ',' hi
 ',' face ',' cold ',' majest ',' contempt ',' but ']
```

3. Lemmatization

Return to table of contents

In computational linguistics, lematization is an algorithmic process of determining the word lemma based on the assumed meaning. Unlike stemming, lemmatization depends on the correct definition of the intended part of speech and the meaning of a word in a sentence, as well as within the broader context surrounding that sentence, such as adjacent sentences or even an entire document. As a result, the development of effective lemmatization algorithms is an open area of research.

In [20]:

```
# NLP model in English spacy library

nlp = spacy . load ( 'en_core_web_sm' )
```

In [21]:

```
# convert text to words with advanced properties (lemmas, POS)

doc = nlp ( '' . join ( no_stop_tokens ))
print ( doc [ 0 : 40 ])
```

```
Amerigo Bonasera sat New York Criminal Court Number 3 waited justice vengeance men c
ruelly hurt daughter tried dishonor The judge formidably heavyfeatured man rolled sl
eeves black robe physically chastise two young men standing bench His face cold maje
stic contempt But
```

In [22]:

```
lemmas = [ token . lemma_ for token in doc ]
print ( lemmas [ 0 : 25 ])
```

```
['Amerigo', 'Bonasera', 'sit', 'New', 'York', 'Criminal', 'Court', 'Number', '3', 'w
ait', 'justice', 'vengeance', ' man ',' cruelly ',' hurt ',' daughter ',' try ',' di
shonor ',' the ',' judge ',' formidably ',' heavyfeature ',' man ',' roll ',' sleeve
' ]
```

4. Counting words

Return to table of contents

Let's turn a collection of text documents into a matrix for counting markers. If the lemmas are more accurate, let's use them as a sign for counting.

In [23]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer ()
X = vectorizer . fit_transform ( lemmas )
X
```

Out [23...

```
<103924x8144 sparse matrix of type '<class' numpy.int64 '>'
        with 93090 stored elements in Compressed Sparse Row format>
```

In [24]:

```
len ( vectorizer . get_feature_names ())
```

Out [24...

```
8144
```

In [25]:

```
print ( vectorizer . get_feature_names () [ 40 : 70 ])
```

```
['31st', '32', '35th', '48th', '4b', '5000', '55th', '90', '96th', 'a22', 'abandon',
'abbandanda', ' abbandando ',' abbandandos', 'abbandundo', 'ability', 'able', 'abor
t', 'abortion', 'abortionist', 'about', 'aboveboard', 'abreast', 'abruptly', 'absenc
e' , 'absently', 'absentminde', 'absentminded', 'absentmindedly', 'absolute']
```

In [26]:

```
sum_words = X . sum ( axis = 0 )
words_freq = [( word , sum_words [ 0 , idx ]) for word , idx in vectorizer
words_freq = sorted ( words_freq , key = lambda x : x [ 1 ], reverse = True )
words_freq[ 0 : 15 ]
```

Out [26...

```
[('he', 1685),
 ('say', 1446),
 ('the', 1017),
 ('don', 941),
 ('man', 883),
 ('michael', 819),
 ('would', 771),
 ('corleone', 747),
 ('get', 737),
 ('go', 724),
 ('make', 607),
 ('hagen', 599),
 ('know', 545),
 ('come', 529),
 ('it', 505)]
```

5. Name Entity Recognition

Return to table of contents

Named Entity Recognition (NER) is probably the first step in retrieving information that seeks to classify and classify named entities in text by predefined categories, such as names of individuals, organizations, locations, time expressions, quantities, monetary values, and percentages. etc. NER is used in many areas of natural language processing (NLP), and it can help answer many real-world questions, such as:

- Which companies were mentioned in the news article?
- Were these products mentioned in complaints or feedback?
- Does the tweet contain a person's name? Does the tweet contain this person's location?

In [27]:
```
import spacy
from spacy import displacy
from collections import Counter
import en_core_web_sm
nlp = en_core_web_sm . load ()
```

One of the nice things about Spacy is that we only need to apply nlp once, the whole background pipeline will return objects.

In [28]:
```
def _print ( obj , depth ):
    print ( str ( obj ) [: depth ])
```

In [29]:
```
doc = nlp ( text )
_print ([( X . text , X . label_ ) for X in doc . ents ], 150 )
```

```
[('Amerigo Bonasera', 'PERSON'), ('New York Criminal Court', 'ORG'), ('two', 'CARDIN
AL'), ('Amerigo Bonasera', 'PERSON'), ('Amerigo Bonasera) ',' PERSON
```

In [30]:
```
sentences = [ x for x in doc . sents ]
displacy . render ( nlp ( str ( sentences [ 10 : 20 ])), jupyter = True , style =
```

["You acted like wild beasts in a jungle and you are fortunate you did not sexually molest that poor girl or I'd put you behind bars for twenty years **DATE** . ", The judge paused, his eyes beneath impressively thick brows flickered slyly toward the sallow-faced Amerigo Bonasera **PERSON** , then lowered to a stack of probation reports before him., He frowned and shrugged as if convinced against his own natural desire., He spoke again., "But because of your youth, your clean records, because of your fine families, and because the law in its majesty does not seek vengeance, I hereby sentence you to three years **DATE** 'confinement to the penitentiary., Sentence to be suspended. ", Only forty years **DATE** of professional mourning kept the overwhelming frustration and hatred from showing on Amerigo Bonasera's **PERSON** face., His beautiful young daughter was still in the hospital with her broken jaw wired together; and now these two **CARDINAL** animales went free ?, It had all been a farce., He watched the happy parents cluster around their darling sons.]

IV. TF-IDF

Return to table of contents

Let's move on to one of the two main methods of solving our problem - the TF-IDF algorithm.

TF-IDF is a numerical statistic that is used to show how important a word is to a document in a collection or body. The value of tf - idf increases in proportion to the number of occurrences of the word in the document and is compensated by the number of documents in the body that contain the word, which helps to correct the fact that some words appear more often. tf-idf is one of the most popular timing schemes today. A survey conducted in 2015 showed that 83% of text systems of recommendations in digital libraries use tf - idf.

Term Frequency: TF (t) = (Number of times term t appears in a document) / (Total number of terms in the document)

Inverse Document Frequency: IDF (t) = log_e (Total number of documents / Number of documents with term t in it)

Let's mark sentences here instead of words and give weight to these sentences.

In [31]:

```
from  nltk.tokenize  import  sent_tokenize

sentences  =  sent_tokenize ( text )
total_documents  =  len ( sentences )
```

We have performed tokenization as one of the components for successful TF-IDF implementation before. Now let's create a frequency matrix.

In [32]:

```
def  _create_frequency_matrix ( sentences ):
    frequency_matrix  =  {}
    stopWords  =  set ( stopwords . words ( "english" ))
    ps  =  PorterStemmer ()

    for  sent  in  sentences :
        freq_table  =  {}
        words  =  word_tokenize ( sent )
        for  word  in  words :
            word  =  word . lower ()
            word  =  ps . stem ( word )
            if  word  in  stopWords :
                continue

            if  word  in  freq_table :
                freq_table [ word ]  + =  1
            else :
                freq_table [ word ]  =  1

        frequency_matrix [ sent [: 15 ]]  =  freq_table

    return  frequency_matrix
```

We will deduce the created matrix:

In [33]:

```
freq_matrix  =  _create_frequency_matrix ( sentences  =  sentences )

_print ( freq_matrix ,  200 )
```

```
{'Amerigo Bonaser': {'amerigo': 1, 'bonasera': 1, 'never': 1, 'done': 1, 'finer': 1,
'work': 1, ',': 2, ' discharg ': 1,' oblig ': 1,' prepar ': 2,' hi ': 1,' old ': 1,'
friend ': 1,' godfath ': 1,' lovingli '
```

Calculate TF (t) according to the previously described formula:

In [34]:
```
def  _create_tf_matrix ( freq_matrix ):
    tf_matrix  =  {}

    for  sent ,  f_table  in  freq_matrix . items ():
        tf_table  =  {}

        count_words_in_sentence  =  len ( f_table )
        for  word ,  count  in  f_table . items ():
            tf_table [ word ]  =  count  /  count_words_in_sentence

        tf_matrix [ sent ]  =  tf_table

    return  tf_matrix
```

Part of the resulting matrix will look like:

In [35]:
```
tf_matrix  =  _create_tf_matrix ( freq_matrix  =  freq_matrix )

_print ( tf_matrix ,  500 )
```

```
{'Amerigo Bonaser': {'amerigo': 0.05263157894736842, 'bonasera': 0.0526315789473684
2, 'never': 0.05263157894736842, 'done': 0.05263157894736842, 'finer': 0.05263157894
736847, 'work discharg ': 0.05263157894736842,' oblig ': 0.05263157894736842,' prepa
r ': 0.10526315789473684,' hi ': 0.05263157894736842,' old ': 0.05263157894736842,'
 friend ': 0.052631578926364242 Stk #: 0.0526315789473
```

If you compare this table with the table we created in the previous step, you will see that words with the same frequency have the same TF score.

Now count the frequencies of each word in the sentences:

In [36]:
```
def  _create_documents_per_words ( freq_matrix ):
    word_per_doc_table  =  {}

    for  sent ,  f_table  in  freq_matrix . items ():
        for  word ,  count  in  f_table . items ():
            if  word  in  word_per_doc_table :
                word_per_doc_table [ word ]  + =  1
            else :
                word_per_doc_table [ word ]  =  1

    return  word_per_doc_table
```

In [37]:
```
count_doc_per_words  =  _create_documents_per_words ( freq_matrix  =  freq_matrix )

_print ( count_doc_per_words ,  500 )
```

```
{'amerigo': 24, 'bonasera': 59, 'never': 304, 'done': 101, 'finer': 2, 'work': 149,
 ',': 4376, 'discharg': 6, ' oblig ': 4,' prepar ': 32,' hi ': 2174,' old ': 197,' f
riend ': 174,' godfath ': 80,' lovingli ': 1,' mother ': 89,' bride ' : 23, 'wed': 5
2, '.': 10197, 'judg': 22, 'formid': 6, 'heavy-featur': 1, 'man': 478, 'roll': 25,
 'sleev' : 4, 'black': 59, 'robe': 1, 'physic': 30, 'chastis': 2, 'two': 277, 'youn
g': 141, 'men': 282, 'stand': 54 , 'befor': 196, 'bench': 1, 'face': 241, 'wa': 237
3, 'cold': 40,
```

Next, we calculate the IDF for each word and generate a matrix.

IDF (t) = log_e (Total number of documents / Number of documents with term t in it)

In [38]:

```
import  math

def  _create_idf_matrix ( freq_matrix ,  count_doc_per_words ,  total_documents ):
    idf_matrix  =  {}

    for  sent ,  f_table  in  freq_matrix . items ():
        idf_table  =  {}

        for  word  in  f_table . keys ():
            idf_table [ word ]  =  math . log10 ( total_documents  /  float ( count_

        idf_matrix [ sent ]  =  idf_table

    return  idf_matrix
```

The resulting matrix:

In [39]:

```
idf_matrix  =  _create_idf_matrix ( freq_matrix  =  freq_matrix ,  count_doc_per_wor

_print ( idf_matrix ,  500 )
```

```
{'Amerigo Bonaser': {'amerigo': 2.676045494138533, 'bonasera': 2.2854047242079947,
 'never': 1.5733831522413853, 'done': 2.0519353620674967, 'finer': 3.755226740186157
7, 'work': 1.8830704794 discharg ': 3.2781054854664955,' oblig ': 3.45419674452217
7,' prepar ': 2.551106757530233,' hi ': 0.7189971960998633,' old ': 1.76179050968854
62,' friend ': 1.8157074875675394,' godfath '25 '' : 2.106866729205226, 'bride': 2.6
9452
```

Now multiply the values from the matrix and create a new matrix:

In [40]:

```
def  _create_tf_idf_matrix ( tf_matrix ,  idf_matrix ):
    tf_idf_matrix  =  {}

    for  ( sent1 ,  f_table1 ),  ( sent2 ,  f_table2 )  in  zip ( tf_matrix . items

        tf_idf_table  =  {}

        for  ( word1 ,  value1 ),  ( word2 ,  value2 )  in  zip ( f_table1 . items (
                                        f_table2 . items ()):   # here,
            tf_idf_table [ word1 ]  =  float ( value1  *  value2 )

        tf_idf_matrix [ sent1 ]  =  tf_idf_table

    return  tf_idf_matrix
```

The result is given below:

In [41]:

```
tf_idf_matrix  =  _create_tf_idf_matrix ( tf_matrix  =  tf_matrix ,  idf_matrix  =

_print ( tf_idf_matrix ,  500 )
```

```
{'Amerigo Bonaser': {'amerigo': 0.14084449969150173, 'bonasera': 0.1202844591688418
```

```
1, 'never': 0.08280963959165186, 'done': 0.10799659800355245, 'finer': 0.19764351264
13767, '10810 ', '10' discharg ': 0.17253186765613132,' oblig ': 0.1817998286590619
4,' prepar ': 0.268537553424235,' hi ': 0.037841957689466486,' old ': 0.092725816299
39716,' friend ': 0.09556355197724591,' friend ': 0.09556355197724591,' friend ' Stk
#: 0.11088772258974
```

Sentence evaluation differs according to different algorithms. Here we use the evaluation of the words Tf-IDF in the sentence to give weight to the paragraph.

In [42]:

```
def  _score_sentences ( tf_idf_matrix )  ->  dict :

    sentenceValue  =  {}

    for  sent ,  f_table  in  tf_idf_matrix . items ():
        total_score_per_sentence  =  0

        count_words_in_sentence  =  len ( f_table )
        for  word ,  score  in  f_table . items ():
            total_score_per_sentence  + =  score

        sentenceValue [ sent ]  =  total_score_per_sentence  /  count_words_in_sente

    return  sentenceValue
```

This gives a table of sentences and their corresponding assessment:

In [43]:

```
_print ( _score_sentences ( tf_idf_matrix  =  tf_idf_matrix ),  700 )
```

```
{'Amerigo Bonaser': 0.12350496550205792, 'The judge, a fo': 0.12487990365716523, 'Hi
s face was co': 0.25988669204772413, 'But there was s': 0.3070131156115828, '' You a
cted like ': 0.0630846769443123,' Yes thoug ': 0.36204185063490685,' Animals. ': 0.6
904045928938374,' The two young m ': 0.29834937061891087,' The judge went ': 0.49771
31793134315,' He frowned and ': 0.2917556947521934,' He spoke again. ': 0.5929 of ':
0.12332812278068175,' Sentence to be ': 0.1324198680320529,' His beautiful y ': 0.12
3548977564992,' It had all been ': 0.9507527641833936,' He watched the ': 0.17387590
789459198,' Oh, they were a ': 0.3632 black bile, ': 0.1923116832656644
```

Like any generalization algorithm, there can be different ways to calculate the threshold. We calculate the average score of the sentence.

In [44]:

```
def  _find_average_score ( sentenceValue )  ->  int :

    sumValues  =  0
    for  entry  in  sentenceValue :
        sumValues  + =  sentenceValue [ entry ]

    # Average value for each sentence from the original text
    average  =  ( sumValues  /  len ( sentenceValue ))

    return  average
```

We get the average score:

In [45]:

```
sentence_scores  =  _score_sentences ( tf_idf_matrix  =  tf_idf_matrix )

_find_average_score ( sentence_scores )
```

Out [45…
```
0.23576135780997265
```

We now generate a summary of the text:

Algorithm: Select a sentence for generalization if the sentence score exceeds the average score.

In [46]:
```
def _generate_summary ( sentences , sentenceValue , threshold ):
    sentence_count = 0
    summary = ''

    for sentence in sentences :
        if sentence [: 15 ] in sentenceValue and sentenceValue [ sentence [: 15
            summary + = "" + sentence
            sentence_count + = 1

    return summary
```

Finally, we summarize:

In [47]:
```
summary = _generate_summary ( sentences , sentence_scores , 1.5 * _find_averag
_print ( summary , 980 )
```

```
Yes, yes, thought Amerigo Bonasera. Animals. Animals. The judge went on. He spoke ag
ain. It had all been a farce. And he had prospered thereby. If she ever did come hom
e. And paid to see it on the screen. She had misjudged his drunkenness. He sprang ov
er the cocktail table and grabbed her by the throat. He fell on top of her. And she
 was giggling at him. I love your daughter with all respect. The Godfather. A respec
t truly earned. That you, you yourself, proclaim your friendship. His reward? He sli
ghted no one. A maiden could do no more. They were not impressed with her. Don Corle
one had no desire, no intention, of letting his youngest son be killed in the servic
e of a foreign power to himself. He enlisted and fought over the Pacific Ocean. He b
ecame a Captain and won medals. All the guests had arrived. Her Cupid-bow mouth pout
ed to give him an airy kiss. She thought him incredibly handsome. He was elaborately
courteous to her as if they were both actors in a play.
```

This result will help us in the future. And now let's move on to the main stage - the creation of a neural network.

V. Neural network

Return to table of contents

First, download the libraries and create a new dataset:

In [48]:
```
import pandas as pd
import csv
from sklearn.model_selection import train_test_split
```

To form a dataset, we use our own algorithm:

- first delete all tabs and newlines
- then divide the script text into parts and combine each part of the tape into one

In [49]:
```
f = open ( D: \\ backup \\ CA-32 \\ Course (machine learning) \\ Puzo Mario-The Go
data = pd . read_excel ( "D: \\ backup \ CA-32 \\ Course (machine learning) \\ dat

temp = f . read () . splitlines ()
tmp = ''
lst_tmp = lst = []
i = into = 0

for index , line in enumerate ( temp ):
    temp [ index ] = line . strip ( ' \ t ' )

for index , line in enumerate ( temp ):
    lst_tmp . append ( line )

def slicee ( n ):
    for i in range ( 0 , len ( lst_tmp ), n ):
        yield lst_tmp [ i : i + n ]

for i in list ( slicee ( 4 )):
    a = '' . join ( i )
    lst . append ( a )

for index , line in enumerate ( lst [ 3 :]):
    row = pd . DataFrame ({ 'Original' : sentences [ index ], 'Result' : line }
    data = data . append ( row , ignore_index = False )

data . head ()
```

Out [49…

	Original	Result
0	Amerigo Bonasera sat in New York Criminal Cour ...	BONASERA
1	The judge, a formidably heavy-featured man, ro ...	America has made my fortune.
2	His face was cold with majestic contempt.	
3	But there was something false in all this that ...	As he speaks, THE VIEW imperceptibly begins to ...
4	“You acted like the worst kind of degenerates, ...	

Next we will conduct a sentiment analysis.

Return to table of contents

In [50]:
```
from itertools import islice
import matplotlib.pyplot as plt

data . shape
def take ( n , iterable ):
    return list ( islice ( iterable , n ))
```

In [51]:
```
affinity_scores = data [ 'Original' ] . to_dict ()
take ( 5 , affinity_scores . items ())
```

Out [51…
```
[(0,
  'Amerigo Bonasera sat in New York Criminal Court Number 3 and waited for justice;
vengeance on the men who had so cruelly hurt his daughter, who had tried to dishonor
her. '),
 (1,
```

```
  'The judge, a formidably heavy-featured man, rolled up the sleeves of his black ro
be as if to physically chastise the two young men standing before the bench.'),
 (2, 'His face was cold with majestic contempt.'),
 (3,
  'But there was something false in all this that Amerigo Bonasera sensed but did no
t yet understand.'),
 (4,
  '"You acted like the worst kind of degenerates," the judge said harshly.')]
```

In the vocabulary of emotions we have lemmatized, but we want to show the original sentence and the original form of words in the results? How to do it?

Next steps:

- put a unique identifier in each sentence (line)
- make a column for the sentence
- calculate a score for each sentence (line) by converting the word to a lemmatized form for comparison only and save it in a new column
- sort the sentences by points to show the 10 and 10 lowest

In [52]:

```
from nltk import tokenize
sentences = tokenize . sent_tokenize ( "" . join ( core_book ))
sentences [ 5 : 13 ]
```

Out [52…

```
['Yes, yes, thought Amerigo Bonasera.',
 'Animals.',
 'Animals.',
 'The two young men, glossy hair crew cut, scrubbed clean-cut faces composed into hu
mble contrition, bowed their heads in submission.',
 'The judge went on.',
 '' You acted like wild beasts in a jungle and you are fortunate you did not sexuall
y molest that poor girl or I'd put you behind bars for twenty years. " The judge pau
sed, his eyes beneath impressively thick brows flickered slyly toward the sallow-fac
ed Amerigo Bonasera, then lowered to a stack of probation reports before him. ',
 'He frowned and shrugged as if convinced against his own natural desire.',
 'He spoke again.']
```

In [53]:

```
sent_df = pd . DataFrame ( sentences , columns = [ 'sentence' ])
sent_df
```

Out [53…

	sentence
0	Amerigo Bonasera sat in New York Criminal Cour ...
1	The judge, a formidably heavy-featured man, ro ...
2	His face was cold with majestic contempt.
3	But there was something false in all this that ...
4	"You acted like the worst kind of degenerates, ...
...	...
11378	Washed clean of sin, a favored supplicant, she ...
11379	She shifted her body to make her weight less p ...
11380	She emptied her mind of all thought of herself ...

	sentence
11381	Then with a profound and deeply willed desire ...
11382	Thank you for downloading the book in free electric ...

11383 rows × 1 columns

Sometimes there is no predefined function that performs everything we want. Therefore, we define our own function, which is specific to our use case.

In [54]:

```
nlp = spacy . load ( 'en_core_web_sm' )
sentiment_lexicon = affinity_scores

def calculate_sentiment ( text : str = None ) -> float :
    sent_score = 0
    if text :
        sentence = nlp ( text )
        for word in sentence :
            sent_score += sentiment_lexicon . get ( word . lemma_ , 0 )
    return sent_score
```

In our case, we want to evaluate each word in the sentence in a lemmatized form, but calculate the evaluation for the entire original sentence.

In [55]:

```
sent_df [ 'sentiment_value' ] = sent_df [ 'sentence' ] . apply ( calculate_sentime
sent_df [ 'word_count' ] = sent_df [ 'sentence' ] . p . split () . apply ( len )
sent_df [ 'word_count' ] . head ( 10 )

sent_df . sort_values ( by = 'sentiment_value' ) . head ( 3 )
```

Out [55…

	sentence	sentiment_value	word_count
0	Amerigo Bonasera sat in New York Criminal Cour ...	0	31
7583	I want all cooperation with the other Families ...	0	24
7584	I want nothing to break this peace no matter w ...	0	21

Next, let's remove some of the main characters' names to make it easier for the neural network to handle the task.

In [56]:

```
for index1 , row in data . iterrows ():
    a = str ( row [ 'Result' ])
    s = a . strip () . split ()
    for c in s :
        if c . isupper () and len ( s ) ! = 1 :
            data . drop ( index1 , axis = 0 , inplace= True )
            break

print ( 'Converted data:' )
data . head ( 13 )
```

```
Converted data:
```

Out [56…

	Original	Result
0	Amerigo Bonasera sat in New York Criminal Cour ...	BONASERA
1	The judge, a formidably heavy-featured man, ro ...	America has made my fortune.
2	His face was cold with majestic contempt.	
4	“You acted like the worst kind of degenerates, ...	
5	Yes, yes, thought Amerigo Bonasera.	BONASERA
8	The two young men, glossy hair crew cut, scrub ...	taught her never to dishonor her
9	The judge went on.	family. She found a boy friend,
10	“You acted like wild beasts in a jungle and yo ...	not an Italian. She went to the
11	He frowned and shrugged as if convinced agains ...	movies with him, stayed out late.
12	He spoke again.	Two months ago he took her for a
thirteen	“But because of your youth, your clean records ...	drive, with another boy friend.
14	Sentence to be suspended. ” Only forty years of ...	They made her drink whiskey and
15	His beautiful young daughter was still in the ...	then they tried to take advantage

Now let's move on to direct network design based on the AutoModel model.

In [57]:

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy
import tensorflow as tf
import autokeras as ak

X , y = data . iloc [:, 0 ] . to_numpy (), data . iloc [:, 1 ] . to_numpy ()
print ( X . shape , y . shape )

X = X . astype ( 'object' )
y = LabelEncoder () . fit_transform ( y )

X_train , X_test , y_train , y_test = train_test_split ( X , y , test_size =
print ( X_train . Shape , X_test . Shape , y_train . Shape , y_test . Shape )
```

```
(6813,) (6813,)
(4564,) (2249,) (4564,) (2249,)
```

Let's build our model. AutoKeras is quite flexible for the data format. For text, the input must be one-dimensional.

For the purposes of regression, it must be a vector of numerical values. AutoKeras accepts numpy.ndarray.

In [58]:

```
input_node = ak . TextInput ()
output_node = ak . TextBlock ( block_type = "ngram" ) ( input_node )
```

```
output_node  =  ak . ClassificationHead () ( output_node )
clf  =  ak . AutoModel (
    inputs = input_node ,  outputs = output_node ,  overwrite = True ,  max_trials =
)
```

Finally, "capture" the data, and then load the tensorboard to visualize the results.

In [61]:

```
from  keras.callbacks  import  TensorBoard

log_dir  =  "C: \\ Users \\ danie \\ OneDrive \\ Documents \\ vs code \\ logs \\ fit

tensorboard_callback  =  tf . keras . callbacks . TensorBoard ( log_dir = log_dir )

clf . fit ( X_train ,
        y_train ,
        epochs = 2 ,
        callbacks = [ tensorboard_callback ])
```

```
Trial 3 Complete [00h 00m 28s]
val_loss: 6.671507835388184

Best val_loss So Far: 6.664879322052002
Total elapsed time: 00h 01m 29s
INFO: tensorflow: Oracle triggered exit
Epoch 1/2
143/143 [==============================] - 11s 51ms / step - loss: 7.6034 - accurac
y: 0.1888
Epoch 2/2
143/143 [==============================] - 3s 24ms / step - loss: 5.4953 - accuracy:
0.2942
INFO: tensorflow: Assets written to:. \ Auto_model \ best_model \ assets
```

Now let's save and view the statistics of our model.

In [62]:

```
model  =  clf . export_model ()

model . save ( "model_autokeras" ,  save_format = "tf" )

model . summary ()
```

```
INFO: tensorflow: Assets written to: model_autokeras \ assets
Model: "model"
_________________________________________________________________
Layer (type) Output Shape Param #
================================================= ===============
input_1 (InputLayer) [(None,)] 0
_________________________________________________________________
expand_last_dim (ExpandLastD (None, 1) 0
_________________________________________________________________
text_vectorization (TextVect (None, 5000) 0
_________________________________________________________________
dense (Dense) (None, 32) 160032
_________________________________________________________________
batch_normalization (BatchNo (None, 32) 128
_________________________________________________________________
re_lu (ReLU) (None, 32) 0
_________________________________________________________________
dense_1 (Dense) (None, 32) 1056
_________________________________________________________________
batch_normalization_1 (Batch (None, 32) 128
_________________________________________________________________
re_lu_1 (ReLU) (None, 32) 0
```

```
________________________________________________________________
dense_2 (Dense) (None, 2689) 88737
________________________________________________________________
classification_head_1 (Softm (None, 2689) 0
================================================= ===============
Total params: 255,081
Trainable params: 249,953
Non-trainable params: 5,128
________________________________________________________________
```

Let's check log files and visualize the results of the model:

In [72]:

```
% load_ext tensorboard
 % tensorboard --logdir = log_dir
```

```
The tensorboard extension is already loaded. To reload it, use:
  % reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 24128), started 11:22:08 ago. (Use '! Kill 241
28' to kill it.)
```

So, as we can see from the visualization, our accuracy does not exceed 30%, ie we have created only a partial model and, unfortunately, not very successful.

However, paired with the results of tf-idf, will give us the opportunity for further research in the future.

References:

Return to table of contents

1. MA Aizerman, EA Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In Automation and Remote Control, number 25 in Automation and Remote Control, pages 821–837, 1964.
2. Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. Journal of Machine Learning Research, 1: 113–141, 2000.
3. Rie Ando and Tong Zhang. A high-performance semi-supervised learning method for text chunking. In Proc. of the 43rd Annual Meeting of the Association for Computa-tional Linguistics (ACL'05), pages 1–9, Ann Arbor, Michigan, June 2005a. DOI: 10.3115 / 1219840.1219841.
4. Rie Ando and Tong Zhang. A framework for learning predictive structures from multi-ple tasks and unlabeled data. Journal of Machine Learning Research, 6: 1817–1853, 2005b.
5. Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuz-man Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In Proc. of the 54th Annual Meeting of the Association for Computa-tional Linguistics— (Volume 1: Long Papers), pages 2442–2452, 2016. DOI: 10.18653 / v1 / P16-1231.
6. Jan A. Botha and Phil Bluns. Compositional morphology for word representations and language modeling. In Proc. of the 31st International Conference on Machine Learning (ICML), Beijing, China, June 2014.