Building a LSTM Network that Classifies an English Text According to One Linguistic Variant: American or British

Text Classification Using Natural Language Processing

Nicolás Díaz Durana

November 2022

Departamento de Matemáticas

Facultad de Matemáticas e Ingenierías

Fundación Universitaria Konrad Lorenz

Bogotá D.C., Colombia

Table of Contents

Building a LSTM Network that Classifies an English Text According to One Linguistic Variant: American or British

Foreword

Text classification using natural language processing (NLP) is a challenging problem of great interest today (Alias, G., & Cassanelli, 2019). The algorithms used in NLP are usually based on neural networks, this being one of the most cited topics of the twentieth century. However, it is not yet fully understood what happens in the hidden layers of these networks and the reason for their efficiency, widely documented.

Computational linguistics is one of the disciplines that studies the problems covered by natural language processing. Studying neural networks in articulation with linguistics is relevant because it builds new knowledge and prepares the ground for new research in a field that is still young, contributing, moreover, to the integration of seemingly distant disciplines. One of the problems that has aroused the most interest in recent decades is the classification of written texts based on some linguistic characteristic or variant (Beysolow, 2018).

This project aims to build a software capable of classifying an English text according to a particular English variant: whether it is British or American. In particular, the model trains a Long Short Term Memory (LSTM), a type of Recurrent Neural Network. The software was coded entirely in Python. The following technical report showcases the development and results of the project. It is divided into five chapters.

The first chapter explains how the data was obtained and preprocessed, while presenting two word clouds as a visual aid for understanding the data. The second chapter focuses on the building and training of the model, while explaining how tokenization, padding and parameter design was accomplished. The third chapter is centered in the assessment of the performance of the model; it describes the metrics used to this end while developing the concept of confusion matrix; it also shares the obtained results after running 18 experiments with different substring length and number of epochs. The fourth chapter presents the bibliography. Finally, the fifth chapter showcases three appendices: 1) the chart of experiment results, 2) the experiments in detail and 3) the mathematical intuition of Long Short Term Memory (LSTM) networks.

The full code of the project, as well as this technical report and the presentation, can be accessed here:

https://github.com/nykolai-d/LSTM_Brown_LOB

The Corpora

Two working corpora were used: the Brown Corpus, a collection of texts written in American English, and the Lancaster-Oslo/Bergen Corpus, a Collection of texts written in British English.

The Brown Corpus of Standard American English was the first of the modern corpora of English language. It is composed of a little more than one million words from all genres of texts in American English published in 1961, and it is categorized in 15 diverse subthemes. Nowadays, it is considered undersized and, to an extent, out of date. However, the corpus is still used mostly in academia, particularly because it has been imitated by other corpus compilers, for example, the LOB (Lancaster-Oslo/Bergen) corpus of British English and the Kolhapur Corpus of Indian English. The LOB corpus is made of some one million words as well, within texts classified in 15 different categories.

These corpora were chosen for several reasons: a) both are composed of texts written during the 1960s, b) both contain at least 1 million words, and c) both have been widely used in research on computational linguistics with neural networks and NLP (Beagle, 1985).

Building a LSTM Network that Classifies an English Text According to One Linguistic Variant: American or British

1. Data Preprocessing

```
[1]  import nltk
     import matplotlib.pyplot as plt
     import pandas as pd
     import math
     from nltk.corpus import brown
     import numpy as np
```

We begin by importing the preliminary libraries and packages that we will use in our model. NLTK stands for Natural Language Toolkit, a robust platform used to build Python code that works with human language. One package included in NLTK's features is the Brown Corpus (line 5). LOB corpus is not available in NLTK, so it was downloaded from a different open source: the Oxford Text Archive. We also import matplotlib for visualization of the data, numpy for mathematical operations and pandas to organize the analyzed text in the form of data frames.

The first steps focus on the preprocessing of both corpora. This includes removing punctuation, numbers and single letter words. Also, all words are set to lowercase. Finally, the preprocessed corpora are put into lists, which will later be used to build the dataframes to train the model.

1.1 The Brown Corpus

```
[2] nltk.download('brown')
    brown.words()[:10]
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
['The',
 'Fulton',
 'County',
 'Grand',
 'Jury',
 'said',
 'Friday',
 'an',
 'investigation',
 'of']
```

Brown Corpus is downloaded from NLTK.

```
print(len(brown.words()))
```

```
1161192
```

Next, we count the number of words in the corpus. There are 1.161.192 words.

```
[4] from nltk.corpus import stopwords
    nltk.download('stopwords')
    import string

    # Added extra punctuation
    punctuation = ['!','"','#','$','%','&',"'",'(',')','*','+',',','-','.','/',
                   ':',';','<','=','>','?','@','[','\\',']','^','_','`','{','|','}','~','``',"''",'--']

    lower_brown = [x.lower() for x in brown.words()]
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

NLTK contains a package called ‘stopwords,’ which contains words of little importance to the training of a model of this nature. A list of punctuation characters is added. Finally, a new list containing the original words in lowercase is created.

```
print(lower_brown[:10])
print(len(lower_brown))
```

```
['the', 'fulton', 'county', 'grand', 'jury', 'said', 'friday', 'an', 'investigation', 'of']
1161192
```

No filtering has been done yet.

```
[6]  pun_stop = punctuation + stopwords.words('english')
     filter_brown = [x for x in lower_brown if x not in pun_stop]
```

```
[7]  print(filter_brown[:10])

     ['fulton', 'county', 'grand', 'jury', 'said', 'friday', 'investigation', "atlanta's", 'recent', 'primary'
```

A list containing the stopwords and the punctuation characters is created. Using this list as reference, the corpus is filtered. Stopwords and punctuation characters are removed.

```
[8]  filter_brown2 = list(filter(lambda x: x.isalpha() and len(x) > 1, filter_brown))
     # remove numbers and single letter words
```

```
[9]  print(filter_brown2[:10])

     ['fulton', 'county', 'grand', 'jury', 'said', 'friday', 'investigation', 'recent', 'primary', 'election']
```

```
[10] print(len(filter_brown2))

     508631
```

Finally, numbers and single letter words are removed from the corpus. The new, preprocessed corpus contains 508.631 words.

1.2 The LOB Corpus

The same process is repeated with the LOB Corpus, with one difference: since it is not available for download through the NLTK packages, it was downloaded from the Oxford Text Archives and put into a drive folder, where it can be accessed from our code.

```
[11] from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive

[12] !ls "/content/drive/My Drive/LOB"

     lob.txt

[13] f = open('/content/drive/My Drive/LOB/lob.txt')
     lob = f.readlines()


[15] print(lob[:100])

     ['<1 TEXT A1>\n', "'STOP ELECTING LIFE PEERS'\n", 'By TREVOR WILLIAMS\n', '   A MOVE to stop Mr. Gaitskel

[16] lob2 = lob.copy()

[17] lob2

      '4, REGULARS AND NO CALL-UP\n',
      'By H. B. BOYNE,\n',
      'Daily Telegraph Political Correspondent\n',
      '   THE next White Paper on defence, to be published in March,\n',
      'is likely to contain a five-year plan for the three Services. Its aim\n',
      'will be to produce superbly equipped, all-Regular forces of about\n',
      '4, men.\n',
```

The initial preprocess requires the separate texts to be joined and some special characters to be removed:

```
[18] lob2 = [''.join(lob2)]

[19] lob3 = str(lob2[0])

[20] lob3 = lob3.replace("<","")
     lob3 = lob3.replace(">","")
     lob3 = lob3.replace("\n"," ")
```

The string is then casted into a list, which can be submitted to the same preprocess of the Brown Corpus.

```
[22] def string_to_list(string):
         new_list = list(string.split(" "))
         return new_list
```

```
[23] lob4 = string_to_list(lob3)
```

```
[24] lob4[:10]
```

```
['1',
 'TEXT',
 'A1',
 "'STOP",
 'ELECTING',
 'LIFE',
 "PEERS'",
 'By',
 'TREVOR',
 'WILLIAMS']
```

The remaining steps are exactly the same as our American corpus: setting to lowercase and removing stopwords, punctuation, numbers and single letter words:

```
[25] lower_lob = [x.lower() for x in lob4]
```

```
[26] print(lower_lob[:10])
     print(len(lower_lob))
```

```
['1', 'text', 'a1', "'stop", 'electing', 'life', "peers'", 'by', 'trevor', 'williams']
1068614
```

```
[27] filter_lob = [x for x in lower_lob if x not in pun_stop]
```

```
[28] print(filter_lob[:10])
```

```
['1', 'text', 'a1', "'stop", 'electing', 'life', "peers'", 'trevor', 'williams', '']
```

```
[29] filter_lob2 = list(filter(lambda x: x.isalpha() and len(x) > 1, filter_lob))
     # remove numbers and single letter words
```

We finalize by counting the number of words in the preprocess LOB Corpus:

```
[30] print(filter_lob2[:10])
```

```
['text', 'electing', 'life', 'trevor', 'williams', 'move', 'stop', 'gaitskell', 'nominating', 'labour']
```

```
[31] print(len(filter_lob2))
```

```
389754
```

The new, preprocessed corpus contains 389.754 words.

1.3 Converting the Corpora to Dataframes

The first step of this section is critical to the evaluation of the model later in the experiments section of our work. It consists in splitting both the corpora (now in the form of lists) into substrings whose size can vary. This report shows only the case where substrings of 10 words were built. However, in the Results section of this report (chapter 3), details of experiments made with substring of 1, 3, 5, 7, 10 and 20 words are included.

```
[32] def create_substrings(x): # x is a list of words
       listoflists = []
       n = 0
       m = 10
       for i in range(int(len(x)/10)):
         listx = []
         for j in range(n,m):
           listx.append(x[j])
         listoflists.append(listx)
         n += 10
         m += 10
       return listoflists
```

This function takes a list of words and returns a list of substrings containing 10 words each.

```
[33] lob_list = create_substrings(filter_lob2)
     brown_list = create_substrings(filter_brown2)
```

```
[34] len(lob_list)

     38975
```

```
[35] len(brown_list)

     50863
```

Both filtered lists containing our corpora are fed to the function, returning lists of substrings which naturally contain ten times less elements, since they were split into new lists of 10 words each.

```
[38] brown_list[0]
```

```
['fulton',
 'county',
 'grand',
 'jury',
 'said',
 'friday',
 'investigation',
 'recent',
 'primary',
 'election']
```

Now, one element of these new lists is a list in itself, as seen in the code above. Dataframes will be created from these new lists of substrings.

```
[40] df_lob['text'] = lob_list
```

```
[41] df_lob['isBrown'] = 0
```

We want to join all the data in one single dataframe which will serve as the training data for our model. Hence, it is required that this joined dataframe includes a column that allows the model to learn from the classification of the different types of text. In the code above, it can be seen that the LOB Corpus texts are labeled with '0' in the binary column called 'isBrown'.

```
[42] df_lob
```

	text	isBrown
0	[text, electing, life, trevor, williams, move,...	0
1	[life, peers, made, meeting, labour, ps, micha...	0
2	[subject, backed, manchester, though, may, gat...	0
3	[likely, turn, line, labour, ps, opposed, gove...	0
4	[peers, put, forward, believes, house, lords, ...	0
...	...	...

Consequently, the Brown Corpus texts are labeled with a '1':

```
[43] df_brown = pd.DataFrame()
```

```
[44] df_brown['text'] = brown_list
```

```
[45] df_brown['isBrown'] = 1
```

```
[46] df_brown
```

	text	isBrown
0	[fulton, county, grand, jury, said, friday, in...	1
1	[produced, evidence, irregularities, took, pla...	1
2	[committee, charge, election, deserves, praise...	1
3	[conducted, term, jury, charged, fulton, super...	1
4	[investigate, reports, possible, irregularitie...	1
...	...	...

The next step consists in the concatenating of the two dataframes that we have just created into one single dataframe:

```
[47] df = pd.concat([df_lob, df_brown]).reset_index(drop = True)
     df
```

	text	isBrown
0	[text, electing, life, trevor, williams, move,...	0
1	[life, peers, made, meeting, labour, ps, micha...	0
2	[subject, backed, manchester, though, may, gat...	0
3	[likely, turn, line, labour, ps, opposed, gove...	0
4	[peers, put, forward, believes, house, lords, ...	0
...	...	...
89833	[headache, count, compassionately, perelman, r...	1
89834	[dropped, seat, exhaling, looked, across, aisl...	1
89835	[song, living, doll, mistake, bang, wide, chee...	1
89836	[midwestern, lineage, mouth, whose, fault, car...	1
89837	[voluptuous, able, gauge, swift, greedy, glanc...	1

89838 rows × 2 columns

Now, we would like to know how many words our joined dataframe has:

```
[48] # Obtain the total words in the dataset
     list_of_words = []
     for i in df.text:
         for j in i:
             list_of_words.append(j)
```

```
[49] len(list_of_words)

     898380
```

```
[50] # Obtain the total number of unique words
     total_words = len(list(set(list_of_words))) #set, only unique ones
     total_words

     51140
```

This quest has been done in two ways: first, we calculated the total number of words, which corresponds to the value that results from adding the words contained in the preprocessed corpora; and second, we calculated the number of unique words, i.e., the words that appear only once in our joined dataset. This value is worth remembering, 51.140, because it will be particularly important in the training stage of our model.

```
[51] # join the words into a string
     df['text_joined'] = df['text'].apply(lambda x: " ".join(x))
```

```
[52] df
```

	text	isBrown	text_joined
0	[text, electing, life, trevor, williams, move,...	0	text electing life trevor williams move stop g...
1	[life, peers, made, meeting, labour, ps, micha...	0	life peers made meeting labour ps michael foot...
2	[subject, backed, manchester, though, may, gat...	0	subject backed manchester though may gather la...
3	[likely, turn, line, labour, ps, opposed, gove...	0	likely turn line labour ps opposed government ...
4	[peers, put, forward, believes, house, lords, ...	0	peers put forward believes house lords abolish...
...	...	...	...
89833	[headache, count, compassionately, perelman, r...	1	headache count compassionately perelman revuls...
89834	[dropped, seat, exhaling, looked, across, aisl...	1	dropped seat exhaling looked across aisle whol...
89835	[song, living, doll, mistake, bang, wide, chee...	1	song living doll mistake bang wide cheekbones ...
89836	[midwestern, lineage, mouth, whose, fault, car...	1	midwestern lineage mouth whose fault carping p...
89837	[voluptuous, able, gauge, swift, greedy, glanc...	1	voluptuous able gauge swift greedy glance figu...

89838 rows × 3 columns

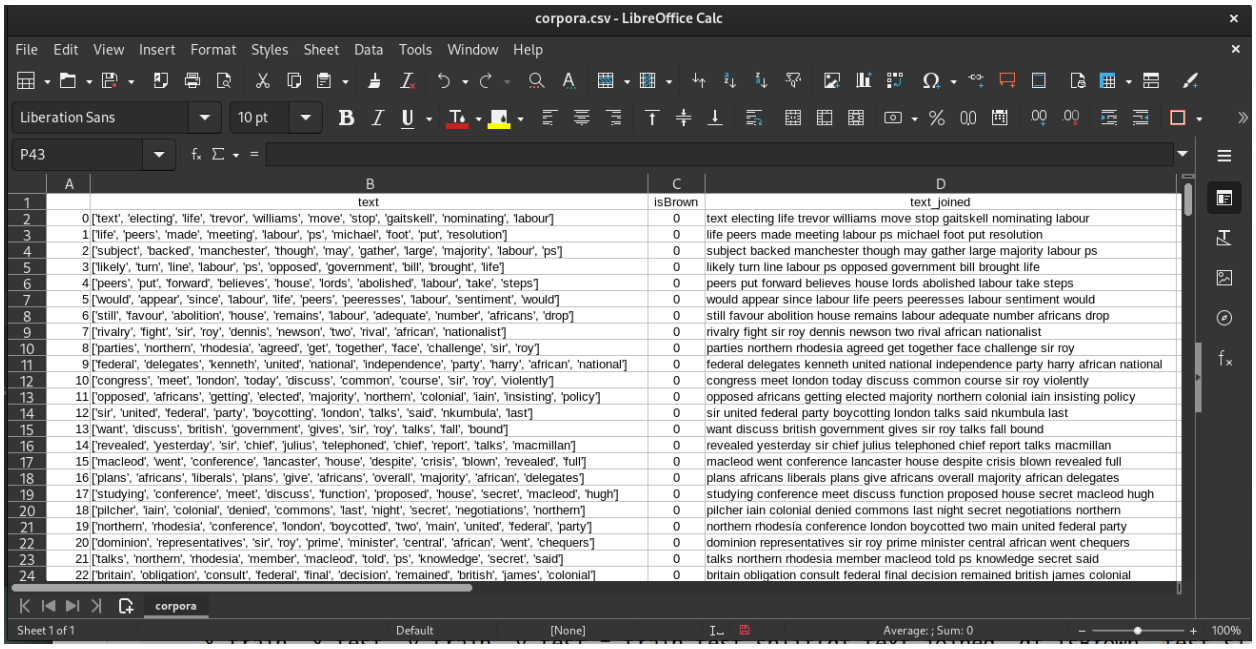
Finally, we join our 10-word lists into 10-word strings separated by spaces (‘ ’) and add the resulting objects to a new column called ‘text_joined’. We will use the data from this column for visualization and model training purposes.

1.4 Visualizing the Cleaned-Up Dataset

It is worth exporting the dataframe with the preprocessed and classified corpora to a .csv file.

```
[53] data = df.to_csv('corpora.csv')
```

This is how our backed-up dataset looks like:



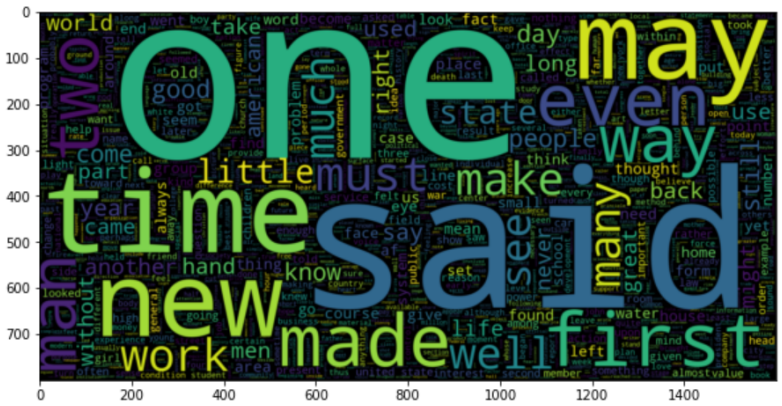
	A	B	C	D
1		text	isBrown	text_joined
2	0	['text', 'electing', 'life', 'trevor', 'williams', 'move', 'stop', 'gaitskell', 'nominating', 'labour']	0	text electing life trevor williams move stop gaitskell nominating labour
3	1	['life', 'peers', 'made', 'meeting', 'labour', 'ps', 'michael', 'foot', 'put', 'resolution']	0	life peers made meeting labour ps michael foot put resolution
4	2	['subject', 'backed', 'manchester', 'though', 'may', 'gather', 'large', 'majority', 'labour', 'ps']	0	subject backed manchester though may gather large majority labour ps
5	3	['likely', 'turn', 'line', 'labour', 'ps', 'opposed', 'government', 'bill', 'brought', 'life']	0	likely turn line labour ps opposed government bill brought life
6	4	['peers', 'put', 'forward', 'believes', 'house', 'lords', 'abolished', 'labour', 'take', 'steps']	0	peers put forward believes house lords abolished labour take steps
7	5	['would', 'appear', 'since', 'labour', 'life', 'peers', 'peeresses', 'labour', 'sentiment', 'would']	0	would appear since labour life peers peeresses labour sentiment would
8	6	['still', 'favour', 'abolition', 'house', 'remains', 'labour', 'adequate', 'number', 'africans', 'drop']	0	still favour abolition house remains labour adequate number africans drop
9	7	['rivalry', 'fight', 'sir', 'roy', 'dennis', 'newson', 'two', 'rival', 'african', 'nationalist']	0	rivalry fight sir roy dennis newson two rival african nationalist
10	8	['parties', 'northern', 'rhodesia', 'agreed', 'get', 'together', 'face', 'challenge', 'sir', 'roy']	0	parties northern rhodesia agreed get together face challenge sir roy
11	9	['federal', 'delegates', 'kenneth', 'united', 'national', 'independence', 'party', 'harry', 'african', 'national']	0	federal delegates kenneth united national independence party harry african national
12	10	['congress', 'meet', 'london', 'today', 'discuss', 'common', 'course', 'sir', 'roy', 'violently']	0	congress meet london today discuss common course sir roy violently
13	11	['opposed', 'africans', 'getting', 'elected', 'majority', 'northern', 'colonial', 'iain', 'insisting', 'policy']	0	opposed africans getting elected majority northern colonial iain insisting policy
14	12	['sir', 'united', 'federal', 'party', 'boycotting', 'london', 'talks', 'said', 'nkumbula', 'last']	0	sir united federal party boycotting london talks said nkumbula last
15	13	['want', 'discuss', 'british', 'government', 'gives', 'sir', 'roy', 'talks', 'fall', 'bound']	0	want discuss british government gives sir roy talks fall bound
16	14	['revealed', 'yesterday', 'sir', 'chief', 'julius', 'telephoned', 'chief', 'report', 'talks', 'macmillan']	0	revealed yesterday sir chief julius telephoned chief report talks macmillan
17	15	['macleod', 'went', 'conference', 'lancaster', 'house', 'despite', 'crisis', 'blown', 'revealed', 'full']	0	macleod went conference lancaster house despite crisis blown revealed full
18	16	['plans', 'africans', 'liberals', 'plans', 'give', 'africans', 'overall', 'majority', 'african', 'delegates']	0	plans africans liberals plans give africans overall majority african delegates
19	17	['studying', 'conference', 'meet', 'discuss', 'function', 'proposed', 'house', 'secret', 'macleod', 'hugh']	0	studying conference meet discuss function proposed house secret macleod hugh
20	18	['pilcher', 'iain', 'colonial', 'denied', 'commons', 'last', 'night', 'secret', 'negotiations', 'northern']	0	pilcher iain colonial denied commons last night secret negotiations northern
21	19	['northern', 'rhodesia', 'conference', 'london', 'boycotted', 'two', 'main', 'united', 'federal', 'party']	0	northern rhodesia conference london boycotted two main united federal party
22	20	['dominion', 'representatives', 'sir', 'roy', 'prime', 'minister', 'central', 'african', 'went', 'chequers']	0	dominion representatives sir roy prime minister central african went chequers
23	21	['talks', 'northern', 'rhodesia', 'member', 'macleod', 'told', 'ps', 'knowledge', 'secret', 'said']	0	talks northern rhodesia member macleod told ps knowledge secret said
24	22	['britain', 'obligation', 'consult', 'federal', 'final', 'decision', 'remained', 'british', 'james', 'colonial']	0	britain obligation consult federal final decision remained british james colonial

Next, using wordcloud, a Python library for building word cloud visualizations, we explore both our corpora:

```
[54] from wordcloud import WordCloud, STOPWORDS
```

```
[55] # plot the word cloud for the Brown Corpus
     plt.figure(figsize = (10,10))
     wc = WordCloud(max_words = 2000 , width = 1600,
                    height = 800).generate(" ".join(df[df.isBrown == 1].text_joined))
     plt.imshow(wc, interpolation = 'bilinear')
```

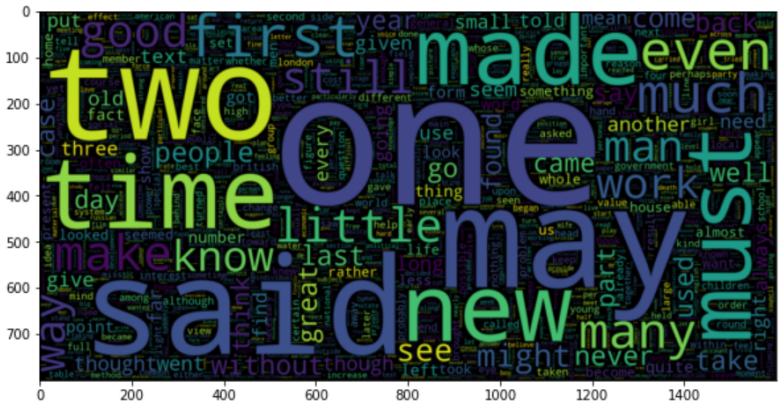
```
<matplotlib.image.AxesImage at 0x7fd118ff8c50>
```



Looking at this word cloud, it is worth saying that this work, when revisited in the future, could consider a more rigorous preprocess (for example, involving stemming), as well as other filtering methods.

```
[56] # plot the word cloud for the LOB Corpus
     plt.figure(figsize = (10,10))
     wc = WordCloud(max_words = 2000 , width = 1600,
                    height = 800).generate(" ".join(df[df.isBrown == 0].text_joined))
     plt.imshow(wc, interpolation = 'bilinear')
```

```
<matplotlib.image.AxesImage at 0x7fd118feffd0>
```



At first view, there doesn't seem to be enormous differences between the two corpora. We have yet to see whether the model can learn from the data in order to correctly classify a text as either American or British.

2. The Model

2.1 Tokenization and Padding

An intermediate step between our preprocess stage and the training of the model is tokenization and padding. Tokenizer features from the library NLTK, as well as the package 'punkt', are used in the process.

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
[58] from nltk.tokenize import word_tokenize, sent_tokenize
```

Tokenization consists in converting a string of natural language into numerical symbols, meaning a sentence (or group of words) will be turned into a vector of numbers, called 'tokens'. This is what the model takes as reference to learn.

Padding is a process that takes these vectors of numbers and makes them uniform in size by adding the '0' value at the end of certain vectors. To this end, we need to know the length of the longest string in the 'text_joined' column of our dataframe.

```
[59] # length of maximum document will be needed to create word embeddings
     maxlen = -1
     for doc in df.text_joined:
         tokens = nltk.word_tokenize(doc)
         if(maxlen<len(tokens)):
             maxlen = len(tokens)
     print("The maximum number of words in any document is =", maxlen)
```

```
The maximum number of words in any document is = 12
```

Next step is to split the data into a test batch and a train batch. It is common practice to allot 80% of the data to train and 20% to train.

```
[59] # split data into test and train
     from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test = train_test_split(df.text_joined, df.isBrown, test_size = 0.2)
```

Tensorflow and Keras are Python libraries popularly used for the building of Machine Learning models. For the tokenizing and padding, they feature the packages 'Tokenizer' and 'pad_sequences'.

```
[ ] from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Embedding, LSTM, Bidirectional
```

This chunk of code also imports the Keras package 'LSTM', which will be used for the training of the model using Long Short Term Memory Networks. Other relevant packages for this step of the process are 'Dense', which serves as a regular densely-connected NN layer; 'Embedding', which converts positive integers (indexes) into dense vectors of fixed size; 'Sequential', which groups a linear stack of layers into a Keras model, later used in the training step.

Finally, we import the Keras package 'Bidirectional', which plays an important role in the building of the model. Its online documentation reads as follows:

> *For sequences other than time series (e.g. text), it is often the case that a RNN model can perform better if it not only processes sequence from start to end, but also backwards. For example, to predict the next word in a sentence, it is often useful to have the context around the word, not only just the words that come before it.* [1]

Hence, we use a bidirectional recurrent neural network (RNN) because it enhances the predictive power of the model.

[1] Bidirectional RNNs: taken from https://www.tensorflow.org/guide/keras/rnn#bidirectional_rnns on November 2022.

```
[61] # Create a tokenizer to create sequences of tokenized words
     tokenizer = Tokenizer(num_words = total_words)
     tokenizer.fit_on_texts(x_train)
     train_sequences = tokenizer.texts_to_sequences(x_train)
     test_sequences = tokenizer.texts_to_sequences(x_test)
```

We now create our sequences of tokenized words using 'Tokenizer'.

```
[62] # Example of tokenized text
     print("The encoding for document\n",df.text_joined[100],"\n is : ",train_sequences[100])

     The encoding for document
      represented sir roderick taken part detailed common market exchanges past
      is :  [139, 1965, 387, 4900, 22403, 1211, 29223, 416, 83, 1616]
```

As an example, this chunk of code prints the text in the 101st position of the column 'text_joined' of our dataframe, followed by its encoded sequence.

```
[64] for i,doc in enumerate(padded_train[:3]):
         print("The padded encoding for document",i+1," is : ",doc)

     The padded encoding for document 1  is :  [  247   273 18505  5862 10591    98   273  1119  5863   561     0     0]
     The padded encoding for document 2  is :  [ 1040   115 22381  1433  2092  9859    59  6362  5864  4473     0     0]
     The padded encoding for document 3  is :  [ 7667  1866   553  9208   574 15946  2406    21  1195 29190     0     0]
```

We proceed to do the padding step, where the aim is to guarantee that all sequences are of length 12 (as calculated above in cell 59) by adding 0 values at the end of each sequence.

2.2 Building the Model

Before the training begins, the parameters of the model are defined:

```
[65] # Sequential Model
     model = Sequential()

     # embeddidng layer
     model.add(Embedding(total_words, output_dim = 128))

     # Bi-Directional RNN and LSTM
     model.add(Bidirectional(LSTM(128)))

     # Dense layers
     model.add(Dense(128, activation = 'relu'))
     model.add(Dense(1,activation= 'sigmoid'))
     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
     model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 128)         6545920

 bidirectional (Bidirectiona  (None, 256)              263168
 l)

 dense (Dense)               (None, 128)               32896

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 6,842,113
Trainable params: 6,842,113
Non-trainable params: 0
_________________________________________________________________
```

Firstly, our model has one input layer and one output one, in the form of sequences of maximum 12 words and a binary classification (0 or 1), respectively. Appropriately, the Tensorflow package 'Sequential' is used for *"a plain stack of layers where each layer has exactly one input tensor and one output tensor"*[2].

Secondly, we introduce an embedding layer, while setting as 128 the number of low-dimensional features we want to represent in the input data. This means that the layer will learn to represent the whole of our Corpora with 128 variables only. This process is somewhat analogous to what takes place in Principal

[2] The Sequential Model: from https://www.tensorflow.org/guide/keras/sequential_model#when_to_use_a_sequential_model on November 2022

Component Analysis and conveys one desirable consequence: that subsequent layers learn more effectively with less compute resources. This step aims to optimize the learning process of the model.

Thirdly, we set our model as 'Bidirectional RNN and LSTM', which means that the model will sequence the inputs from start to finish –and also backwards–, while remembering long term dependencies by default. This aims to build a model capable of recalling information for a prolonged period of time.

Next, we set the dense layers with the following two activation functions:

- 'relu': the rectified linear unit activation function, which returns max(x, 0), the element-wise maximum of 0 and the input tensor.

$$f(x) = x^+ = \max(0, x)$$

- 'sigmoid': the sigmoid activation function, given by sigmoid(x) = 1 / (1 + exp(-x)). This is the "classifier" function, which will assign a value close to zero for small values (<-5), and a value close to 1 for large values (>5).

$$S(x) = \frac{1}{1 + e^{-x}}$$

As an optimization method we choose the algorithm 'adam', a stochastic gradient descent method. According to Kingma et al. (2014), this algorithm is "*computationally efficient, has little memory requirement (...) and is well suited for problems that are large in terms of data/parameters*".

Finally, we set our loss function as 'binary_crossentropy', which is ideal for binary (0 or 1) classification models, such as ours, for it calculates the cross-entropy loss between true labels and predicted labels. The label 'acc' in the metrics category refers to 'accuracy'.

2.3 Training the Model

We now proceed to training our model:

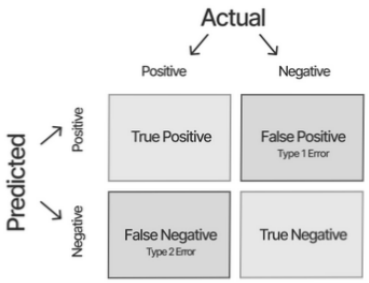
```
[67] y_train = np.asarray(y_train)

[68] # train the model, substrings = 10
     model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 1)

     1011/1011 [==============================] - 147s 141ms/step - loss: 0.5416 - acc: 0.7150 - val_loss: 0.4775 - val_acc: 0.7619
     <keras.callbacks.History at 0x7f465201fd10>
```

Here, we reproduced the code with the parameters that had the best performance of all the experiments that were run: training over substrings of 10 words throughout 1 single epoch. Other results with different levels of accuracy are presented in the next section of this report.

3. Results

3.1 The Confusion Matrix and the Assessment Metrics

The metrics precision, accuracy, recall and F1-score are used to evaluate our model. These metrics are intimately linked to the concept of confusion matrix.



Confusion Matrix

A confusion matrix is a tool that serves the purpose of mapping data, in a way that it can be known where the model is getting "confused". In other words, it allows us to calculate the *accuracy* and *precision* of the model.

In a binary class dataset, where there only exist two categories of data, it is common to name the data as either "positive" and "negative". Generally, the confusion matrix will allow us to identify the data as follows:

- *True Positive (TP)*: the positive class being classified correctly.
- *True Negative (TN)*:the negative class being classified correctly.
- *False Positive (FP)*: the negative class being classified wrongly.
- *False Negative (FN)*: the positive class but being classified wrongly.

Hence, the metrics that we are interested in for the assessment of our model can be gathered from a binary class confusion matrix, as follows:

Accuracy. The number of samples classified correctly.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision. The number of samples from to the positive class out of those that were actually *predicted* to be of the positive.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall. The number of samples predicted correctly to be belonging to the positive class out of all the samples that *actually belong* to the positive class.

$$Recall = \frac{\#\ of\ True\ Positives}{\#\ of\ True\ Positives + \#\ of\ False\ Negatives}$$

F1-Score. The F1 score is defined as the harmonic mean of precision and recall. It is recommended for classification ML models.

$$\text{F1-Score} = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

After the building of the model, 18 experiments were run.

3.2 The Model's Performance

To evaluate the model's performance, we begin by asking the model to make a series of predictions:

```
[69] # make prediction
     pred = model.predict(padded_test)

     562/562 [==============================] - 10s 17ms/step
```

To this end, we use the in-build function of the Tensorflow library, 'model.predict'. The function receives the 'padded_test' object, which was built in cell 63:

```
[63] padded_train = pad_sequences(train_sequences,maxlen = 12, padding = 'post', truncating = 'post')
     padded_test = pad_sequences(test_sequences,maxlen = 12, truncating = 'post')
```

Next, we iterate through the data gathered in the previous step and classify the predictions of the model into a list called ‘prediction’:

```
[70] # if the predicted value is >0.5 it is American (Brown), else it is British (LOB)
     prediction = []
     for i in range(len(pred)):
         if pred[i].item() > 0.5:
             prediction.append(1)
         else:
             prediction.append(0)
```

Then, we use the in-built function ‘accuracy_score’ from sklearn.metrics, a scikit-learn Python package, to assess the accuracy of this particular prediction:

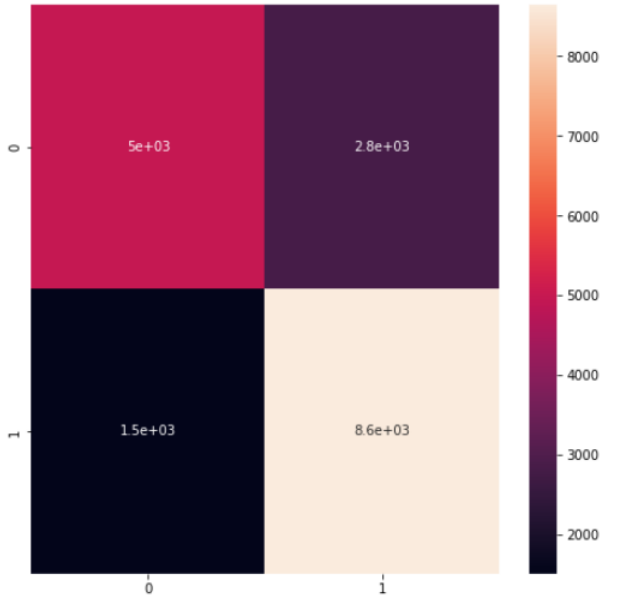
```
[71] from sklearn.metrics import accuracy_score

     accuracy = accuracy_score(list(y_test), prediction)

     print("Model Accuracy : ", accuracy)

     Model Accuracy :  0.7585707925200356
```

Using the visualization Python, we can get and print the confusion matrix of the model's prediction:

```
[72] import seaborn as sns
```

```
# get the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(list(y_test), prediction)
plt.figure(figsize = (8, 8))
sns.heatmap(cm, annot = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f464be68b10>
```



Finally, we compute the metrics that interest us, using in-built functions from sklearn.metrics:

```
[74] # category dict
     category = { 0: 'British', 1 : "American"}
```

```
[75] from sklearn.metrics import precision_score, recall_score, f1_score

     precision = precision_score(y_test, prediction)
     recall = recall_score(y_test, prediction)
     f1 = f1_score(y_test, prediction)

     print("Precision:", precision)
     print("Recall:", recall)
     print("F1 score:", f1)
```

```
Precision: 0.7534843205574913
Recall: 0.851545579838551
F1 score: 0.7995193640817082
```

As a reference, we establish the classification labels through the creation of a dictionary. It is worth noting

that this particular model (substrings=10, epochs=1) had a fairly good performance. The other model that proved worthy had substrings=20 and epoch=1.

4. Bibliography

- Alias, G., & Cassanelli, R. (2019). NLP aplicado a análisis de texto (Doctoral dissertation, Universidad Nacional de Mar del Plata. Facultad de Ingeniería. Argentina).
- Ayata, D., Saraclar, M., & Özgür, A. (2017). Busem at semeval-2017. Sentiment analysis with word embedding and long short term memory RNN approaches. In Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017) (pp. 777-783).
- Azevedo, A., & Santos, M. F. (2008). KDD, SEMMA and CRISP-DM: a parallel overview. IADS-DM.
- Beale, A. D. (1985). Grammatical analysis by computer of the Lancaster-Oslo/Bergen (LOB) corpus of British English texts. In the 23rd Annual Meeting of the Association for Computational Linguistics (pp. 293-298).
- Beysolow II, T. (2018). Applied Natural Language Processing with Python: Implementing Machine Learning and Deep Learning Algorithms for Natural Language Processing. Apress.
- Gensim- Documentation. En https://pypi.org/project/gensim/ Retrieved: June 2022
- Hardeniya, N., Perkins, J., Chopra, D., Joshi, N., & Mathur, I. (2016). Natural language processing: python and NLTK. Packt Publishing Ltd.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.
- Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Krohn, J., Beyleveld, G., & Bassens, A. (2019). Deep learning illustrated: a visual, interactive guide to artificial intelligence. Addison-Wesley Professional.
- Kucera, H., & Francis, W. N. (1967). The Brown University standard corpus of present day American English.
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
- NLTK - Documentation. En https://www.nltk.org/ Retrieved: June 2022
- Powers, D. M. (2020). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. arXiv preprint arXiv:2010.16061.
- Štajner, S., & Mitkov, R. (2011, September). Diachronic stylistic changes in British and American varieties of 20th century written English language. In Proceedings of the Workshop on Language Technologies for Digital Humanities and Cultural Heritage (pp. 78-85).

- Suresh, A (2020). What is a confusion matrix? En Analytics Vidhya https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5 Retrieved: June 2022
- Tensorflow - Documentation. En https://www.tensorflow.org/ Retrieved: June 2022

5. Appendices

Appendix 1: Chart of Experiment Results

Substrings	Epochs	Model Accuracy	Precision	Recall	F1 Score
1	1	0.56568175	0.56557044	1	0.72251037
	3	0.56557044	0.56557044	1	0.72251037
	6	0.56569844	0.56511087	0.99084825	0.72238587
3	1	0.44172106	0.51415481	0.31333764	0.38937891
	3	0.61770156	0.64602448	0.71981615	0.68092696
	6	0.4809744	0.624847	0.21056508	0.31498457
5	1	0.58467831	0.58731647	0.89353013	0.70876343
	3	0.54825478	0.59825478	0.59704797	0.59765076
	6	0.50498107	0.56136275	0.5707257	0.56600551
7	1	0.69931432	0.70189443	0.81277593	0.75327664
	3	0.64679756	0.69196804	0.67883663	0.68533944
	6	0.63312295	0.65716387	0.73700495	0.69479824
10	1	0.76319011	0.7809831	0.80868625	0.79459328
	3	0.72579029	0.77695716	0.72359241	0.74932587
	6	0.71822128	0.76762837	0.72240737	0.74433166
20	1	0.75790293	0.75232741	0.84845483	0.79750488
	3	0.74109528	0.75757002	0.79298732	0.77487417
	6	0.73541852	0.77743526	0.74512507	0.76093734

Appendix 2: Experiments

Substrings: 1, epochs: 1

```
# train the model, substrings = 1
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 1)
```

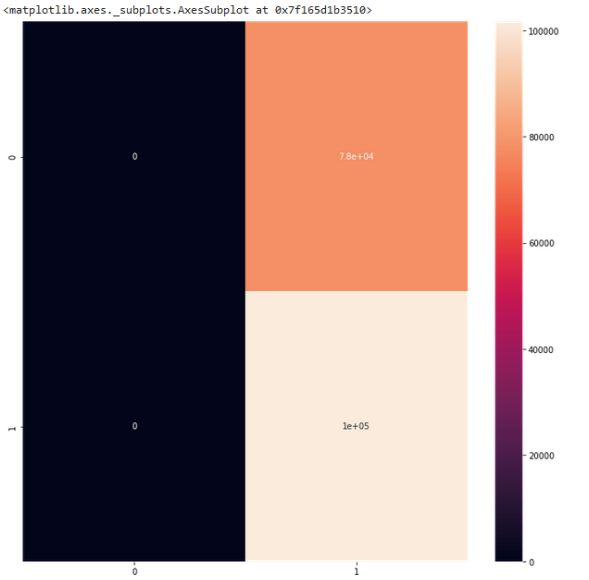
```
10107/10107 [==============================] - 1177s 116ms/step - loss: 0.6626 - acc: 0.5886 - val_loss: 0.6536 - val_acc: 0.5999
<keras.callbacks.History at 0x7f165d625510>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.5656817511423277
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.5656817511423277
Recall: 1.0
F1 score: 0.7226011936711967
```



Substrings: 1, epochs: 3

```
# train the model, substrings = 1
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 3)
```

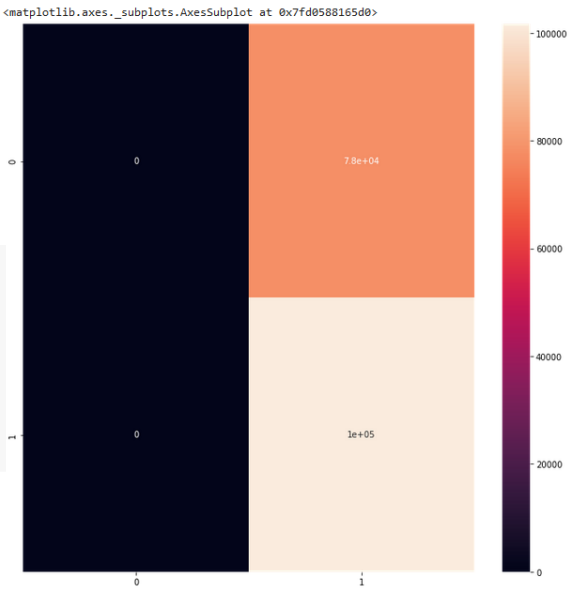
```
Epoch 1/3
10107/10107 [==============================] - 1310s 129ms/step - loss: 0.6629 - acc: 0.5883 - val_loss: 0.6521 - val_acc: 0.6004
Epoch 2/3
10107/10107 [==============================] - 1325s 131ms/step - loss: 0.6307 - acc: 0.6250 - val_loss: 0.6546 - val_acc: 0.5997
Epoch 3/3
10107/10107 [==============================] - 1355s 134ms/step - loss: 0.6218 - acc: 0.6289 - val_loss: 0.6567 - val_acc: 0.6014
<keras.callbacks.History at 0x7fd058ad79d0>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.5655704402900761
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.5655704402900761
Recall: 1.0
F1 score: 0.7225103716001238
```



Substrings: 1, epochs: 6

```
# train the model, substrings = 1
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 6)
```

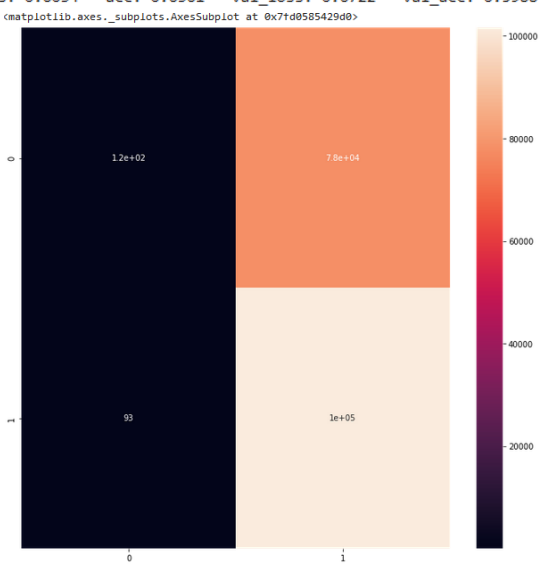
```
Epoch 1/6
10107/10107 [==============================] - 1328s 131ms/step - loss: 0.6146 - acc: 0.6318 - val_loss: 0.6553 - val_acc: 0.6004
Epoch 2/6
10107/10107 [==============================] - 1321s 131ms/step - loss: 0.6104 - acc: 0.6335 - val_loss: 0.6594 - val_acc: 0.6019
Epoch 3/6
10107/10107 [==============================] - 1339s 133ms/step - loss: 0.6078 - acc: 0.6346 - val_loss: 0.6626 - val_acc: 0.6019
Epoch 4/6
10107/10107 [==============================] - 1343s 133ms/step - loss: 0.6058 - acc: 0.6356 - val_loss: 0.6698 - val_acc: 0.5999
Epoch 5/6
10107/10107 [==============================] - 1342s 133ms/step - loss: 0.6044 - acc: 0.6361 - val_loss: 0.6655 - val_acc: 0.6009
Epoch 6/6
10107/10107 [==============================] - 1338s 132ms/step - loss: 0.6034 - acc: 0.6361 - val_loss: 0.6722 - val_acc: 0.5988
<keras.callbacks.History at 0x7fd05856d5d0>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.5656984477701653
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.5657108788196225
Recall: 0.9990848258216887
F1 score: 0.7223858720400728
```



Substrings: 3, epochs: 1

```
# train the model substrings = 3
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 1)
```

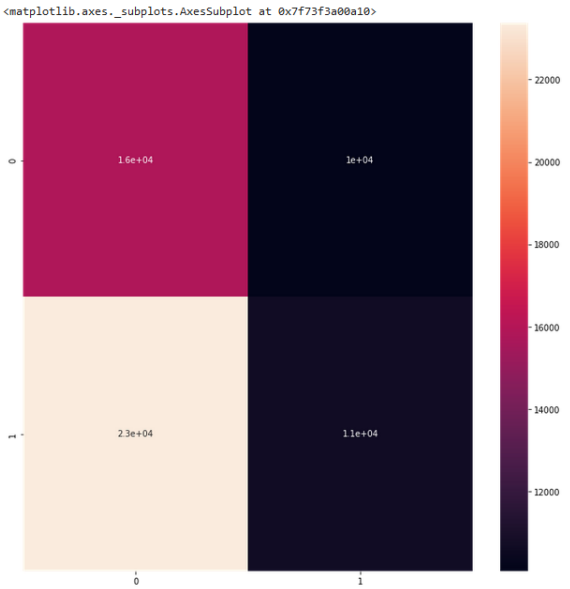
```
3369/3369 [==============================] - 556s 164ms/step - loss: 0.6257 - acc: 0.6358 - val_loss: 0.5946 - val_acc: 0.6653
<keras.callbacks.History at 0x7f740aaefed0>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.4417210692401449
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.5141548107065348
Recall: 0.31333764401598874
F1 score: 0.389378914881572
```



Substrings: 3, epochs: 3

```
# train the model substrings = 3
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 3)
```

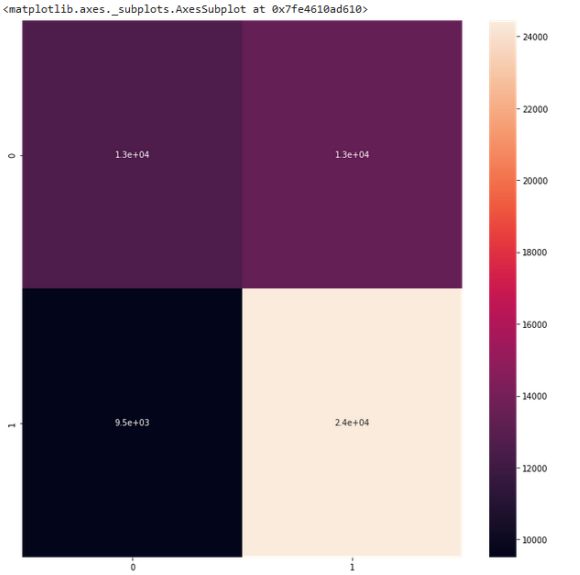
```
Epoch 1/3
3369/3369 [==============================] - 516s 152ms/step - loss: 0.6257 - acc: 0.6344 - val_loss: 0.5986 - val_acc: 0.6642
Epoch 2/3
3369/3369 [==============================] - 498s 148ms/step - loss: 0.5454 - acc: 0.7044 - val_loss: 0.5960 - val_acc: 0.6656
Epoch 3/3
3369/3369 [==============================] - 499s 148ms/step - loss: 0.4861 - acc: 0.7403 - val_loss: 0.6393 - val_acc: 0.6543
<keras.callbacks.History at 0x7fe4611221d0>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.6177015677959027
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.6460244850471984
Recall: 0.719816156973661
F1 score: 0.6809269659006981
```



Substrings: 3, epochs: 6

```
# train the model substrings = 3
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 6)
```

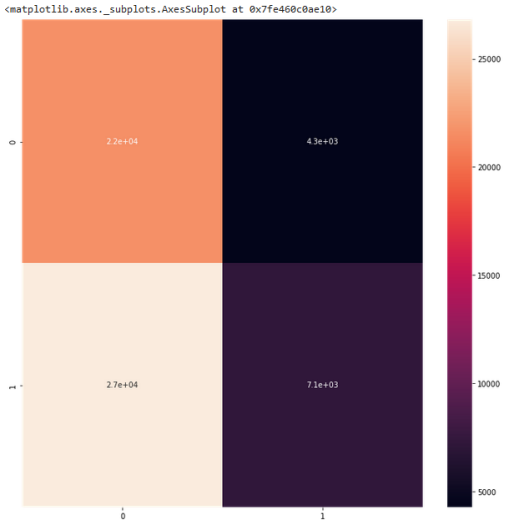
```
Epoch 1/6
3369/3369 [==============================] - 502s 149ms/step - loss: 0.4239 - acc: 0.7754 - val_loss: 0.6982 - val_acc: 0.6497
Epoch 2/6
3369/3369 [==============================] - 491s 146ms/step - loss: 0.3650 - acc: 0.8068 - val_loss: 0.7970 - val_acc: 0.6370
Epoch 3/6
3369/3369 [==============================] - 502s 149ms/step - loss: 0.3145 - acc: 0.8328 - val_loss: 0.9536 - val_acc: 0.6295
Epoch 4/6
3369/3369 [==============================] - 516s 153ms/step - loss: 0.2704 - acc: 0.8574 - val_loss: 1.1619 - val_acc: 0.6257
Epoch 5/6
3369/3369 [==============================] - 514s 153ms/step - loss: 0.2326 - acc: 0.8782 - val_loss: 1.3893 - val_acc: 0.6265
Epoch 6/6
3369/3369 [==============================] - 513s 152ms/step - loss: 0.1990 - acc: 0.8964 - val_loss: 1.5049 - val_acc: 0.6258
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.48097440435443206
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.6248470012239902
Recall: 0.21056508160980497
F1 score: 0.3149845747025121
```



```
# train the model, substrings = 5
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 1)
```

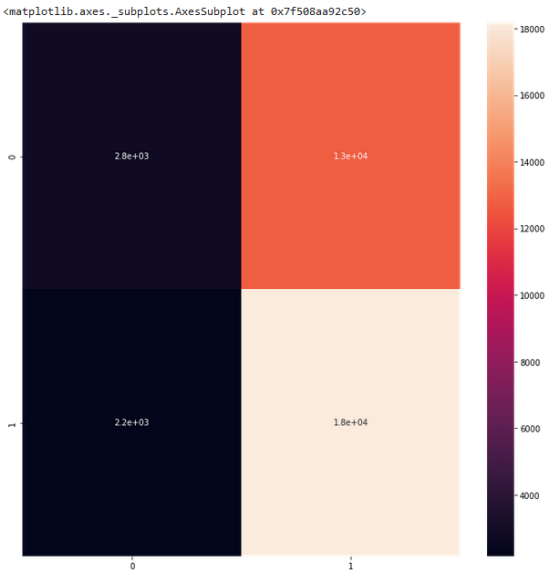
```
2022/2022 [==============================] - 265s 129ms/step - loss: 0.5946 - acc: 0.6683 - val_loss: 0.5544 - val_acc: 0.7018
<keras.callbacks.History at 0x7f508aeeb510>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.5846783170080142
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.5873164737080396
Recall: 0.8935301353013531
F1 score: 0.7087634398111109
```



Substrings: 5, epochs: 3

```
# train the model, substrings = 5
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 3)
```

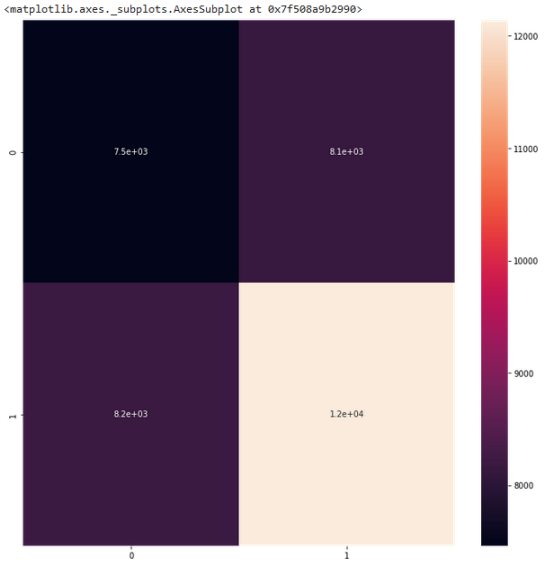
```
Epoch 1/3
2022/2022 [==============================] - 259s 128ms/step - loss: 0.4721 - acc: 0.7591 - val_loss: 0.5566 - val_acc: 0.7024
Epoch 2/3
2022/2022 [==============================] - 259s 128ms/step - loss: 0.3820 - acc: 0.8091 - val_loss: 0.6246 - val_acc: 0.6914
Epoch 3/3
2022/2022 [==============================] - 258s 128ms/step - loss: 0.2918 - acc: 0.8582 - val_loss: 0.7472 - val_acc: 0.6693
<keras.callbacks.History at 0x7f50ed482050>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.5453305877114871
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.5982547820942615
Recall: 0.5970479704797048
F1 score: 0.5976507670713389
```



Substrings: 5, epochs: 6

```
# train the model, substrings = 5
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 6)
```

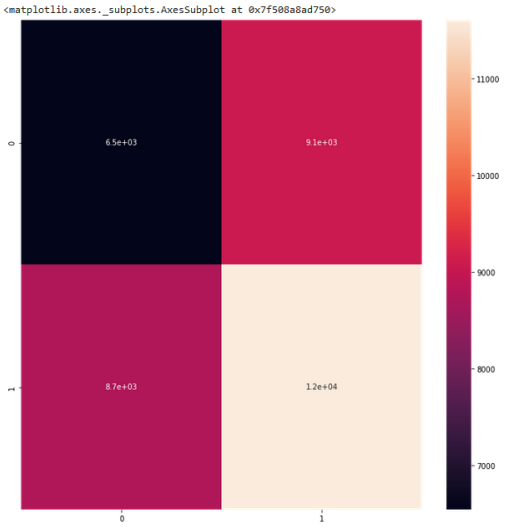
```
Epoch 1/6
2022/2022 [==============================] - 258s 128ms/step - loss: 0.2156 - acc: 0.8962 - val_loss: 1.0272 - val_acc: 0.6802
Epoch 2/6
2022/2022 [==============================] - 255s 126ms/step - loss: 0.1572 - acc: 0.9247 - val_loss: 1.1492 - val_acc: 0.6655
Epoch 3/6
2022/2022 [==============================] - 254s 126ms/step - loss: 0.1144 - acc: 0.9460 - val_loss: 1.5268 - val_acc: 0.6692
Epoch 4/6
2022/2022 [==============================] - 262s 130ms/step - loss: 0.0833 - acc: 0.9608 - val_loss: 1.8360 - val_acc: 0.6602
Epoch 5/6
2022/2022 [==============================] - 271s 134ms/step - loss: 0.0615 - acc: 0.9717 - val_loss: 2.1820 - val_acc: 0.6624
Epoch 6/6
2022/2022 [==============================] - 268s 133ms/step - loss: 0.0475 - acc: 0.9785 - val_loss: 2.4081 - val_acc: 0.6586
<keras.callbacks.History at 0x7f508aa28490>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.5049810774710597
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.5613627564847077
Recall: 0.5707257072570726
F1 score: 0.5660055136744004
```



Substrings: 7, epochs: 1

```
[69] # train the model, substrings = 7
     model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 1)
```

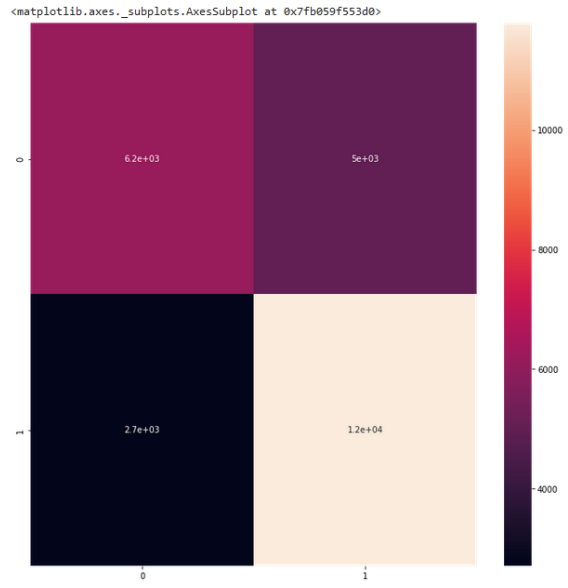
```
1444/1444 [==============================] - 171s 115ms/step - loss: 0.5718 - acc: 0.6892 - val_loss: 0.5192 - val_acc: 0.7336
<keras.callbacks.History at 0x7fb05e4c6110>
```

```
[72] from sklearn.metrics import accuracy_score

     accuracy = accuracy_score(list(y_test), prediction)

     print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.6993143213339567
```

```
[76] from sklearn.metrics import precision_score, recall_score, f1_score

     precision = precision_score(y_test, prediction)
     recall = recall_score(y_test, prediction)
     f1 = f1_score(y_test, prediction)

     print("Precision:", precision)
     print("Recall:", recall)
     print("F1 score:", f1)
```

```
Precision: 0.7018944358393899
Recall: 0.8127759381898455
F1 score: 0.7532766447158111
```



Substrings: 7, epochs: 3

```
# train the model, substrings = 7
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 3)
```

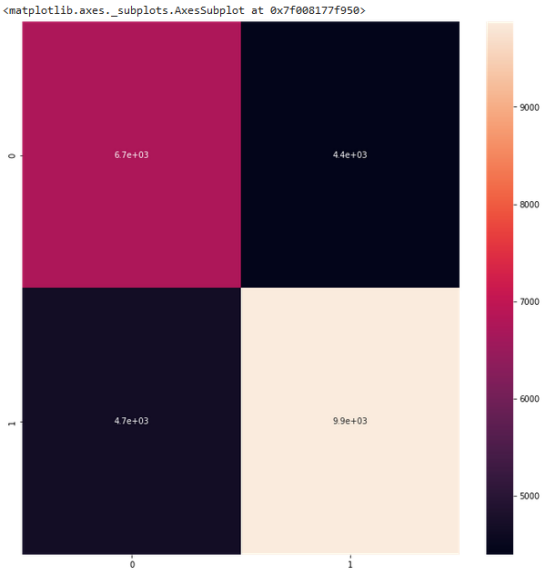
```
Epoch 1/3
1444/1444 [==============================] - 199s 135ms/step - loss: 0.5719 - acc: 0.6882 - val_loss: 0.5203 - val_acc: 0.7326
Epoch 2/3
1444/1444 [==============================] - 190s 132ms/step - loss: 0.4159 - acc: 0.7984 - val_loss: 0.5342 - val_acc: 0.7308
Epoch 3/3
1444/1444 [==============================] - 190s 131ms/step - loss: 0.3018 - acc: 0.8583 - val_loss: 0.6161 - val_acc: 0.7228
<keras.callbacks.History at 0x7f008569fd50>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.6467975689574568
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.6919680403700589
Recall: 0.6788366336633663
F1 score: 0.6853394418992086
```



Substrings: 7, epochs: 6

```
# train the model, substrings = 7
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 6)
```

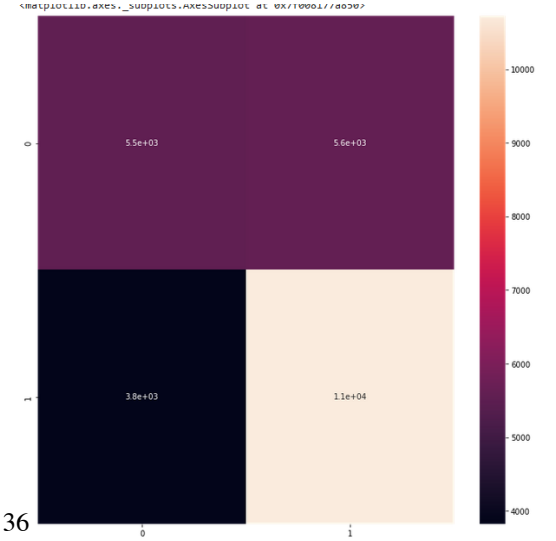
```
Epoch 1/6
1444/1444 [==============================] - 190s 131ms/step - loss: 0.2022 - acc: 0.9068 - val_loss: 0.8098 - val_acc: 0.7170
Epoch 2/6
1444/1444 [==============================] - 189s 131ms/step - loss: 0.1275 - acc: 0.9425 - val_loss: 1.1082 - val_acc: 0.7129
Epoch 3/6
1444/1444 [==============================] - 188s 130ms/step - loss: 0.0756 - acc: 0.9677 - val_loss: 1.3382 - val_acc: 0.7043
Epoch 4/6
1444/1444 [==============================] - 189s 131ms/step - loss: 0.0436 - acc: 0.9825 - val_loss: 1.6967 - val_acc: 0.7005
Epoch 5/6
1444/1444 [==============================] - 190s 131ms/step - loss: 0.0268 - acc: 0.9898 - val_loss: 2.0456 - val_acc: 0.6926
Epoch 6/6
1444/1444 [==============================] - 190s 131ms/step - loss: 0.0164 - acc: 0.9941 - val_loss: 2.2346 - val_acc: 0.6938
<keras.callbacks.History at 0x7f00851ec410>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.6331229546517064
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.6571638771381276
Recall: 0.7370049504950495
F1 score: 0.6947982498784638
```



Substrings: 10, epochs: 1

```
# train the model, substrings = 10
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 1)
```

```
1011/1011 [==============================] - 129s 123ms/step - loss: 0.5431 - acc: 0.7153 - val_loss: 0.4791 - val_acc: 0.7619
<keras.callbacks.History at 0x7fcff92aea50>
```

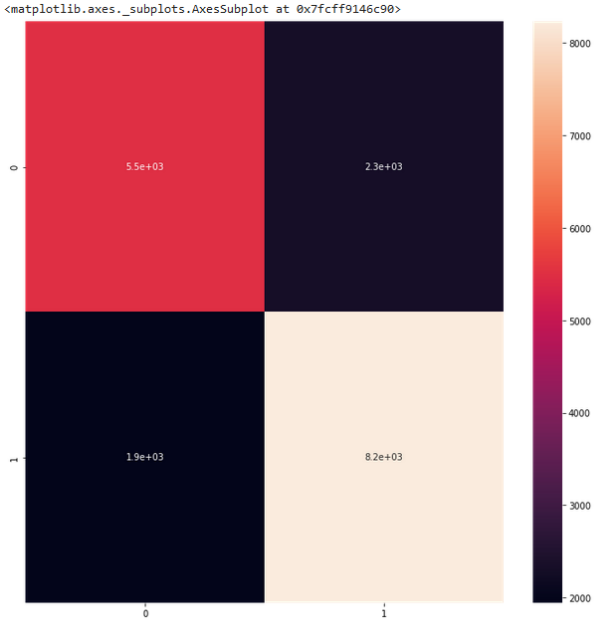
```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.7631901157613535
```

```
[76] from sklearn.metrics import precision_score, recall_score, f1_score

    precision = precision_score(y_test, prediction)
    recall = recall_score(y_test, prediction)
    f1 = f1_score(y_test, prediction)

    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 score:", f1)
```

```
Precision: 0.7809831087492883
Recall: 0.8086862533163015
F1 score: 0.7945932898865556
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff9146c90>
```



Substrings: 10, epochs: 3

```
# train the model, substrings = 10
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 3)
```

```
Epoch 1/3
1011/1011 [==============================] - 124s 123ms/step - loss: 0.3516 - acc: 0.8380 - val_loss: 0.4944 - val_acc: 0.7597
Epoch 2/3
1011/1011 [==============================] - 124s 123ms/step - loss: 0.2248 - acc: 0.8996 - val_loss: 0.6243 - val_acc: 0.7582
Epoch 3/3
1011/1011 [==============================] - 123s 122ms/step - loss: 0.1228 - acc: 0.9481 - val_loss: 0.8781 - val_acc: 0.7381
<keras.callbacks.History at 0x7fcffd7e6e90>
```

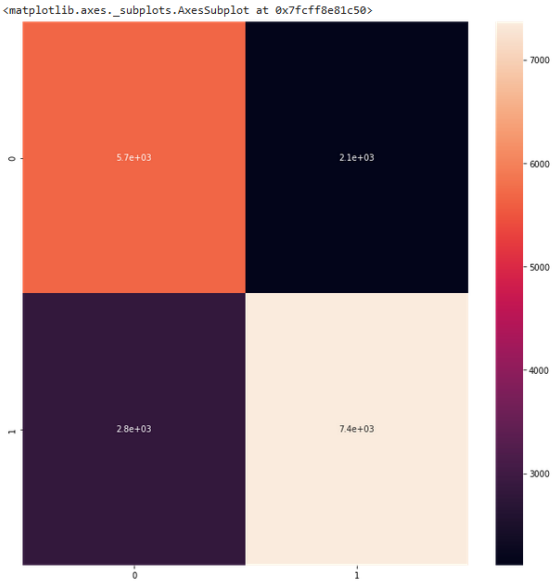
```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.7257902938557436
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.7769571639586411
Recall: 0.7235924142674659
F1 score: 0.7493258712795727
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcff8e81c50>
```



Substrings: 10, epochs: 6

```
[ ] # train the model, substrings = 10
    model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 6)
```

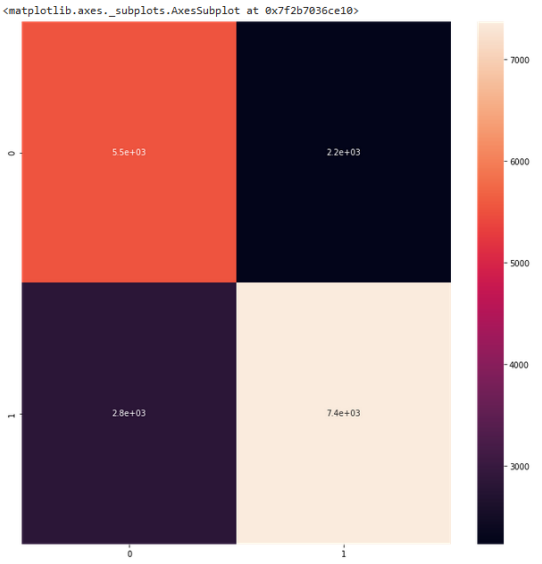
```
Epoch 1/6
1011/1011 [==============================] - 84s 80ms/step - loss: 0.5426 - acc: 0.7129 - val_loss: 0.4763 - val_acc: 0.7605
Epoch 2/6
1011/1011 [==============================] - 78s 78ms/step - loss: 0.3517 - acc: 0.8376 - val_loss: 0.4961 - val_acc: 0.7681
Epoch 3/6
1011/1011 [==============================] - 78s 77ms/step - loss: 0.2236 - acc: 0.9014 - val_loss: 0.5915 - val_acc: 0.7514
Epoch 4/6
1011/1011 [==============================] - 83s 82ms/step - loss: 0.1225 - acc: 0.9478 - val_loss: 0.8825 - val_acc: 0.7515
Epoch 5/6
1011/1011 [==============================] - 81s 80ms/step - loss: 0.0593 - acc: 0.9771 - val_loss: 1.0715 - val_acc: 0.7427
Epoch 6/6
1011/1011 [==============================] - 82s 81ms/step - loss: 0.0278 - acc: 0.9898 - val_loss: 1.4557 - val_acc: 0.7330
<keras.callbacks.History at 0x7f2b706eaa50>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.7182212822796082
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.7676283720445787
Recall: 0.7224073711037051
F1 score: 0.7443316669191536
```



Substrings: 20, epochs: 1

```
[69] # train the model, substrings = 20
     model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 1)
```

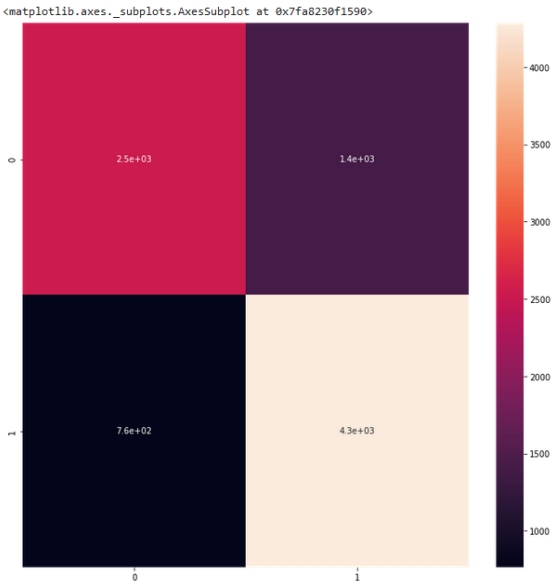
```
506/506 [==============================] - 78s 145ms/step - loss: 0.5595 - acc: 0.7014 - val_loss: 0.5020 - val_acc: 0.7543
<keras.callbacks.History at 0x7fa8233fe290>
```

```
[72] from sklearn.metrics import accuracy_score

     accuracy = accuracy_score(list(y_test), prediction)

     print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.7579029385574354
```

```
[76] from sklearn.metrics import precision_score, recall_score, f1_score

     precision = precision_score(y_test, prediction)
     recall = recall_score(y_test, prediction)
     f1 = f1_score(y_test, prediction)

     print("Precision:", precision)
     print("Recall:", recall)
     print("F1 score:", f1)
```

```
Precision: 0.7523274196381521
Recall: 0.8484548335974643
F1 score: 0.7975048878130527
```



Substrings: 20, epochs: 3

```
# train the model, substrings = 20
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 3)
```

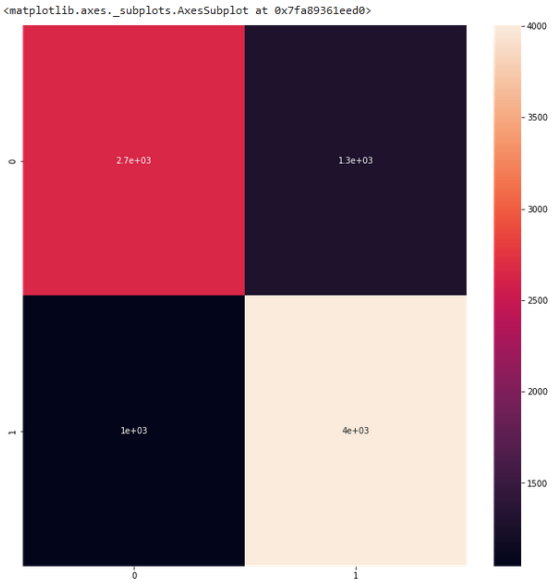
```
Epoch 1/3
506/506 [==============================] - 74s 146ms/step - loss: 0.3086 - acc: 0.8656 - val_loss: 0.5695 - val_acc: 0.7493
Epoch 2/3
506/506 [==============================] - 68s 134ms/step - loss: 0.1366 - acc: 0.9458 - val_loss: 0.7242 - val_acc: 0.7359
Epoch 3/3
506/506 [==============================] - 68s 134ms/step - loss: 0.0508 - acc: 0.9806 - val_loss: 1.2074 - val_acc: 0.7315
<keras.callbacks.History at 0x7fa8935000d0>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.7410952804986642
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

```
Precision: 0.7575700227100681
Recall: 0.7929873217115689
F1 score: 0.7748741773132017
```



Substrings: 20, epochs: 6

```
# train the model, substrings = 20
model.fit(padded_train, y_train, batch_size = 64, validation_split = 0.1, epochs = 6)
```

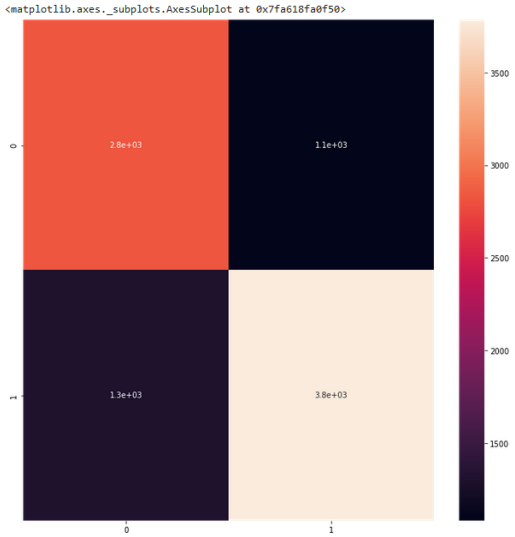
```
Epoch 1/6
506/506 [==============================] - 75s 138ms/step - loss: 0.5631 - acc: 0.6998 - val_loss: 0.4873 - val_acc: 0.7551
Epoch 2/6
506/506 [==============================] - 70s 138ms/step - loss: 0.3124 - acc: 0.8623 - val_loss: 0.5052 - val_acc: 0.7538
Epoch 3/6
506/506 [==============================] - 68s 134ms/step - loss: 0.1492 - acc: 0.9386 - val_loss: 0.7402 - val_acc: 0.7357
Epoch 4/6
506/506 [==============================] - 68s 134ms/step - loss: 0.0622 - acc: 0.9764 - val_loss: 1.0305 - val_acc: 0.7318
Epoch 5/6
506/506 [==============================] - 71s 140ms/step - loss: 0.0276 - acc: 0.9904 - val_loss: 1.2424 - val_acc: 0.7173
Epoch 6/6
506/506 [==============================] - 71s 139ms/step - loss: 0.0118 - acc: 0.9961 - val_loss: 1.8053 - val_acc: 0.7165
<keras.callbacks.History at 0x7fa6192066d0>
```

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(list(y_test), prediction)

print("Model Accuracy : ", accuracy)
```

```
Model Accuracy :  0.7354185218165628
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, prediction)
recall = recall_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

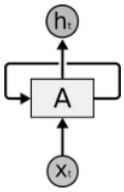
```
Precision: 0.7774352651048089
Recall: 0.7451250738625173
F1 score: 0.7609373428542694
```



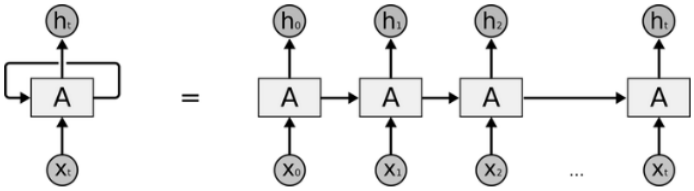
Appendix 3: Mathematical Intuition of Long Short Term Memory Networks

Recurrent Neural Networks

Human understanding of concepts is based on previous knowledge and other preconceptions that allow new interpretations and ideas to be built. In general, artificial neural networks cannot do this. They base their mechanics on the information which is being received and processed in real time, without taking in account any previous information. However, recurrent neural networks do solve this problem. These networks contain loops that allow information to prevail and influence future loops.



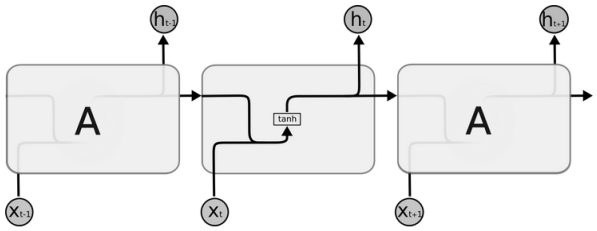
Take the above neural network, A, which takes an input x_t and gives a corresponding value h_t. The loop passes information from one iteration of the network to another. Recurrent neural networks could be understood as various reproductions of itself. Consider the following diagram:



It can be seen how recurrent neural networks are related to chain sequences. One particular, very effective type of neural network is the one known as “LSTMs”.

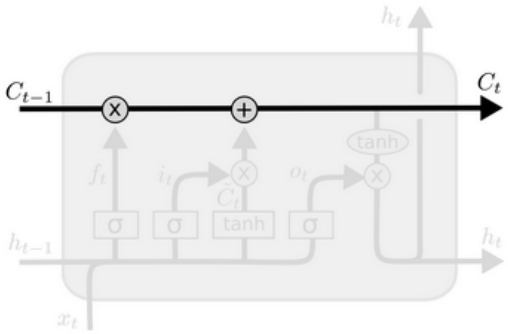
Long Short Term Memory Networks

Long Short Term Memory networks are a special case of recurrent neural networks, which have the possibility of learning long-term dependencies. Usually, in normal recurrent neural networks the repeating module has a simple structure, for example, a *tanh* layer:

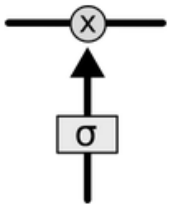


LSTMs are also chain-like, but instead of one single neural network layer, they have four, the cell state becoming the key element of the network.

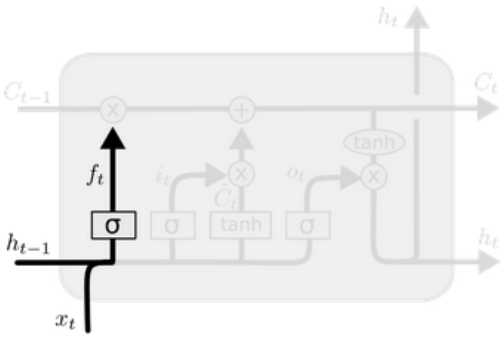
The cell state is the main pipeline of the system. Information traverses the whole chain with few linear interactions and very few changes.



The LSTM is capable of removing or adding information to the cell state. This is regulated by objects called gates, which may or may not let information through, mathematically defined as a sigmoid neural net layer and a pointwise multiplication operation.

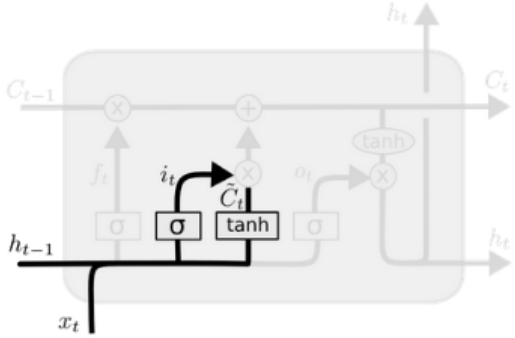


Numbers between zero and one will be spat out by the sigmoid layer outputs, zero being no information at all is passing through and one meaning all information is passing through. This is the first step in the LSTM network. In this point, the model evaluates h_{t-1} and x_t, and returns a value anywhere between 0 and 1 for each initial value in the particular cell state C_{t-1}.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

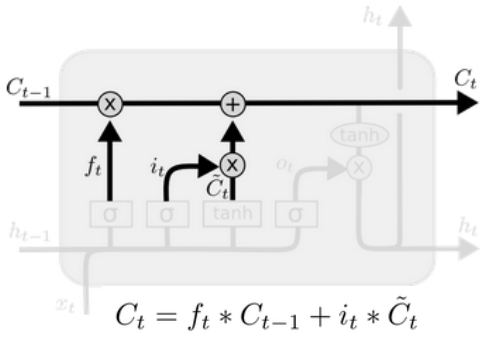
Now there is a sigmoid function layer, called the “input gate layer”, in charge of filtering those values that will be updated. These new values, created by the *tanh* layer, then form a vector that could be added to the state. The combination of these two opens the possibility of producing a new state for the model.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

When the old state is multiplied by f_t, then the old cell state C_{t-1} becomes the new cell state C_t. Here, while previous information is being forgot, new candidate values are created through the operation $i_t * \tilde{C}_t$



At this point, all that's left is to generate an output, which will be derived from the current cell state but will constitute a filtered value in the end. Hence, a sigmoid layer is run, followed by a *tanh* layer, which will push values between -1 and 1. The product of these two will generate the output that we are looking for.