Data Extraction using Semantic Similarity

Arunim Garg, Dhivya Chandrasekaran, Mohiuddin MD Abdul Qudar
Kazi Zainab Khanam

***Abstract*—Natural Language Processing tasks are finding greater significance by Aritifical Intelligence experts over the years and one of the major challenges faced by them is the sparcity of structured text datasets to enable rigorous training of developed models. In this article we propose a data extraction technique that extracts aligned sentence pairs from parallel corpora. Aligned sentences are extracted from the Newsela simplified datasets and simple English Wikipedia where the datasets are used to increase the performance of the auto-encoder sequence-to-sequence text simplification model. The performance is seen to improve with the availability of quality datasets. The major focus of the project is on implementing the semantic similarity algorithm and show the impact of the extracted data on the text simplification algorithm.**

I. Introduction

With the exponential increase in text data generated over time, Natural Language Processing (NLP) has gained significant attention from Artificial Intelligence (AI) experts. Measuring the semantic similarity between various text components like words, sentences, or documents plays a significant role in a wide range of NLP tasks like information retrieval [2], text summarization [3], text classification [4], essay evaluation [5], machine translation [6], question answering [7], [8], among others. One of the major challenges faced by various NLP tasks is the availability of efficient and sufficient data resources to train models in-order to achieve better performances. This paper implements a technique for data extraction using MASSAligner [1] to improve the performance of a simple sequence-to-sequence auto-encoder model for text simplification task. The remaining paper is organised as follows: Section II provides a brief overview on semantic similarity and text simplification tasks. Section III discusses methodology used in the project. Section IV tabulates the results and Section VI concludes with details of proposed future work and limitations.

II. Background

A. Semantic Similarity

Text similarity is assessing the similarity between two words, sentences or expressions, based on the likelihood of how similar is their meaning. There are two predominant methods used for calculating the similarity between expressions, corpus-based or distributional semantic models (DSMs), and knowledge-based models. Knowledge-based methods use word senses, parts of speech and taxonomic information to calculate the similarity between expressions, whereas the corpus-based methods determine similarity based on the assumption that similar words occur in similar documents. The major drawback of corpus-based methods is that they ignore the sentence structure and the difference in word meaning based on the context, while the knowledge-based approach is restricted by the availability of human-crafted dictionaries. Recent researchers have worked on hybrid models combining the above methods to generate similar sentence pairs with better performance than the traditional methods [10], [11]. A clear understanding between 'text-relatedness' and 'text-similarity' is an integral part of text extraction based on similarity. Recent work by [12], [13], uses web search engine results for calculating word relatedness, where words with opposite meanings also have a high similarity score.

B. Text Simplification

Most of the early work in text simplification involved extractive methods of summarization – extracting the sentences that convey the most meaning. These are easiest to implement and available readily online for free. The simplest of all summarization techniques is TF-IDF (Term Frequency – Inverse Document Frequency). This technique involves generating a frequency table for all the words in the corpus, after removing stop words. The sentence weight is then calculated based on the word frequencies, and is normalized by sentence length, to calculate the sentences with the most "substance". We can use a threshold to retain sentences with the most substance or summarize a given text in a given number of sentences by choosing the ones with the highest values. As research in NLP has branched widely over the past decade, primarily due to increased availability of computing resources, the research in simplification has shifted towards abstractive approaches – actual generation of text. Abstractive approaches have mostly focused on lexical or phrasal substitutions for sentence-level simplification. This approach is the simplest to implement as it involves minimal data preprocessing. Truly abstractive simplification approaches involve sentence splitting, text deletion, and addition. One of the approaches used for simplification, and the one we plan to adapt, involves seq2seq modeling, which also used in machine translation. We take aligned complex and simplified sentence pairs, instead of sentences in two languages, convert them to a fixed encoded vector and train an RNN or LSTM to be able to simplify our content. The performance of this approach relies heavily on the dataset being used to train the model.

III. Methodology and Experimental Analysis

This section describes the semantic similarity model, text simplification model, and the two different datasets (Newsela simplified dataset and simple English Wikipedia datasets) used.

A. Datasets

We use two different datasets to provide a comparison of how the performance increase with a bigger dataset and the impact of semantic similarity in building the dataset.

- ***English Wikipedia and Simple English Wikipedia (SEW)***: Simple English Wikipedia is widely used as it is publicly available and because of the popularity of the regular English Wikipedia. SEW includes simplified versions of articles in regular English Wikipedia, and datasets are available with aligning "equivalent" sentences from the two, to allow seq2seq model training.
- ***Newsela Corpus***: This dataset, created by Xu et al. contains news articles with 4 simplified versions for each, produced manually by professional editors. Corpus level simplification is available, however, it has to be processed for sentence-level simplification.

From these datasets, we use the below mentioned Semantic similarity model to extract complex and simple sentence pairs that are then fed to the text simplification autoencoder model to analyze the performance.

B. Semantic similarity Model:

The model we implement to extract semantically similar aligned sentence pairs was proposed by Paetzold et al., The MASSAligner is available as an open-source Python library. The steps involved can be divided into two sections:

1) ***Measuring semantic similarity****:* The model calculates the semantic similarity between sentence pairs using the TF-IDF model. The model converts documents to a bag of words, forms word vectors based on the normalized term frequency (frequency of a term / total number of terms) and finally calculates the cosine similarity between word vectors. The model returns a similarity matrix containing the similarity between all the sentences.

2) ***Aligning sentences based on similarity****:* The model follows a vicinity driven approach to extract the sentences that are similar based on a threshold value provided as a hyperparameter. The aligner then traverses through the similarity matrix to establish an alignment path that searches for the best pair of similar sentences. Given a coordinate $[i, j]$ in a matrix there are three vicinities taken into consideration $V1 = [i, j+1], [i+1, j], [i+1, j+1], V2 = [i+1, j+2], [i+2, j+1]$, and $V3$ as all remaining [x,y] where $x > i$ and $y > j$. The initial point begins at the coordinate that is closest to the first point $(0, 0)$ and has a similarity greater than 0.2 (hyperparameter achieved by [14] (The algorithm searches the three vicinities to find the coordinates having the highest similarity. Thus, the similar pairs are extracted and we write them to two different files.

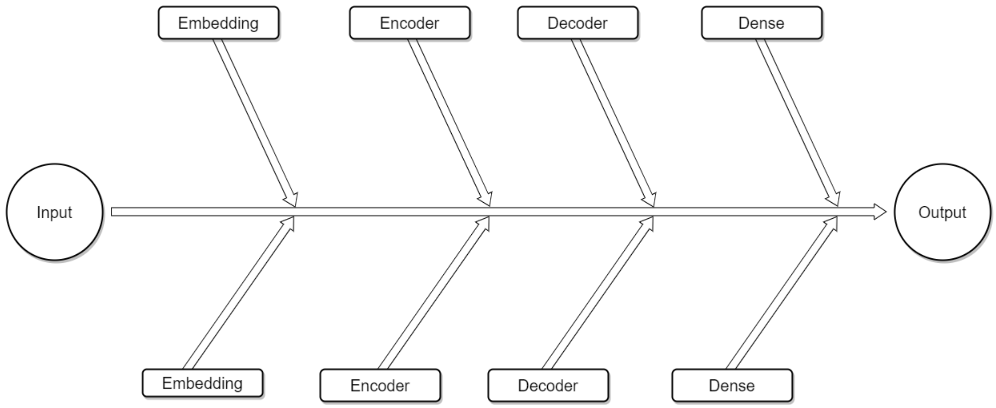

Fig. 1. Architecture of the Auto-encoder Text Simplification Model

C. Text Simplification Model:

Text-simplification is done using a model inspired by a standard machine-translation model called the seq2seq model. This is an auto-encoder model that predicts the simplified version of a complex sentence. The model is simple and at present, uses a simple long short term memory (LSTM) model for encoder/decoder functionalities. The model consists of an embedding layer where the sentences are converted to vectors based on a comprehensive vocabulary built using all the words present in both the simplified and complex documents. The vectors are then passed through two LSTM layers of which one serves as an encoder; encoding the input sequence and producing internal state vectors which serve as conditioning for the decoder. The other would serve as the decoder that is responsible for predicting the target sequence. Figure 1 shows the architecture of the model.

IV. Results

We establish a baseline score using the precompiled Wikipedia dataset that has 167,000 aligned normal and simplified sentence pairs. We extract sentences from the Newsela corpus using the MASSAlign semantic similarity algorithm based on five different semantic similarity thresholds. Five datasets generated by using 30%, 40%, 50%, 60% and 70% similarity percentage between the normal and simplified sentence pairs. Table I shows the number of sentence pairs generated at each semantic similarity thresholds. From the Fig 2, we can see that the number of sentences generated is highest when the similarity threshold is set at 30% and the least number of sentences are generated threshold is set at 70%.

SNo	Percentage Similarity $(\%)$	Number of sentences
1	30	277082
2	40	249200
3	50	217680
4	60	184336
5	70	151588

TABLE I
SENTENCE COUNT COMPARISON AT VARIOUS SIMILARITY LEVELS.

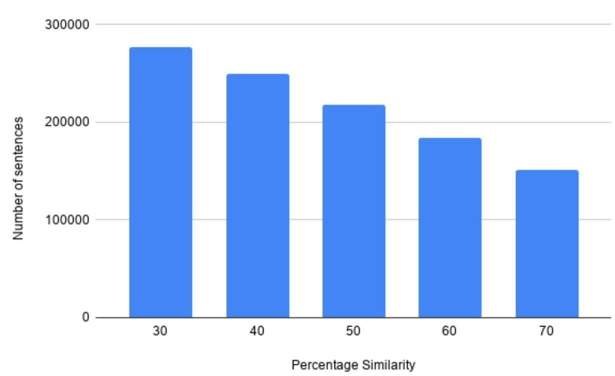

Fig. 2. Comparison of datasets generated using different similarity measures

We then use the Wikipedia based base-line model to perform hyperparamter tuning. We tune the following hyperparameters: sentence length, removing punctuation, removing stop_words, number of

Hyper-parameters	Option 1	Option 2	Option 3	Best Option
Remove punctuation	True	False	–	**True**
Remove stopwords	True	False	–	**False**
Words cutoff	15	20	25	**15**
Aggr length	15	12	10	**15**
Hidden size	128	256	512	**256**
Batch size	128	64	32	**64**

TABLE II
HYPERPARAMETER TUNING.

hidden neurons, and batch_size using the baseline model. Table I shows the hyperparameters that we have tuned our model with and the optimal values of the different hyperparameters.The performance is measured using bilingual evaluation understudy (BLUE) score. The BLUE score is within the range between 0 and 1. The baseline model achieved the best results by removing punctuation, using batch size of 64, hidden size of 512, total aggregated length of 15 words in the simplified sentence and when the word limit(words_cutoff) was set at 15 words as shown in table II.

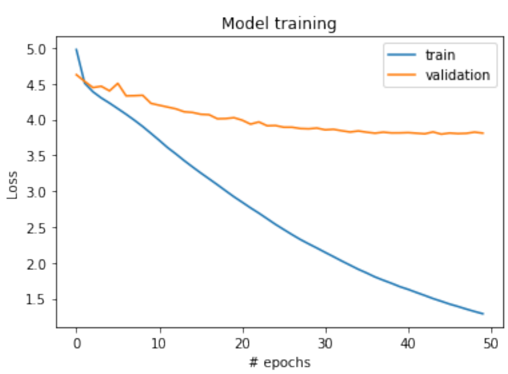

Fig. 3. Training and Validation loss score for our model with 50% sentence similarity with normal and simplified text file

The five different datasets extracted using our semantic similarity algorithm were used to enhance the performance of the base-line model. The performance of the model increases with the use of the Newsela dataset. However, the best results were obtained with the complex and simplified sentence pair dataset extracted at a semantic similarity threshold of 50% as shown in Table III. Fig 3 is a graphical representation of the variation in BLEU score over different datasets. The training and validation graph of the best performance achieved is shown in Fig 4

SNo	Semantic Similarity Threshold	BLEU Score
1	30 % similarity	0.1895
2	40 % similarity	0.2942
3	**50 % similarity**	**0.3134**
4	60 % similarity	0.3128
5	70 % similarity	0.2910

TABLE III
PERFORMANCE OF THE MODEL WITH DIFFERENT DATASETS.

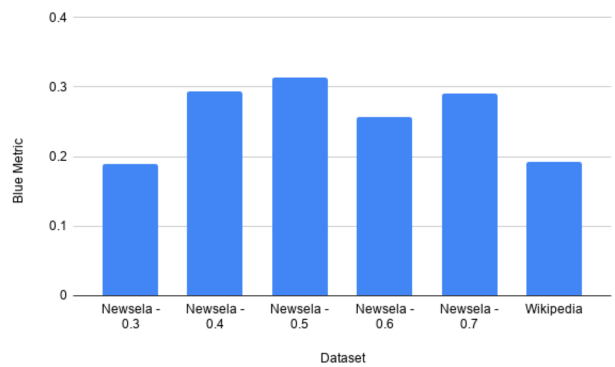

Fig. 4. Comparison of datasets generated using different similarity measures

V. FUTURE WORK

This project has mainly focused on implementing the semantic similarity algorithm to perform data extraction from parallel corpora, hence the text simplification model used is rather simple. However, with better resources and more time, we plan to implement a more complex text simplification model and enhance the performance further.

VI. AUTHORS' CONTRIBUTIONS

- Mohiuddin Md Qudar Abdul - Concept development, implementation and designing figures and tables. 25% percent.
- Arunim Garg - Concept development, implementation and proof reading. 25% percent.

- Dhivya Chandrasekaran - Concept development, implementation and report content development . 25% percent.
- Kazi Zainab - Concept development, implementation and log file compilation and designing figures and tables. 25% percent.

References

[1] Paetzold, Gustavo, Fernando Alva-Manchego, and Lucia Specia. "Massalign: alignment and annotation of comparable documents." In Proceedings of the IJCNLP 2017, System Demonstrations, pp. 1-4. 2017.

[2] Kim, Sun, Nicolas Fiorini, W. John Wilbur, and Zhiyong Lu. "Bridging the gap: Incorporating a semantic similarity measure for effectively mapping PubMed queries to documents." Journal of biomedical informatics 75 (2017): 122-127.

[3] Mohamed, Muhidin, and Mourad Oussalah. "SRL-ESA-TextSum: A text summarization approach based on semantic role labeling and explicit semantic analysis." Information Processing & Management 56, no. 4 (2019): 1356-1372.

[4] Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882 (2014).

[5] Janda, Harneet Kaur, Atish Pawar, Shan Du, and Vijay Mago. "Syntactic, Semantic and Sentiment Analysis: The Joint Effect on Automated Essay Evaluation." IEEE Access 7 (2019): 108486-108503.

[6] Zou, Will Y., Richard Socher, Daniel Cer, and Christopher D. Manning. "Bilingual word embeddings for phrase-based machine translation." In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pp. 1393-1398. 2013.

[7] Bordes, Antoine, Sumit Chopra, and Jason Weston. "Question answering with subgraph embeddings." arXiv preprint arXiv:1406.3676 (2014).

[8] Lopez-Gazpio, Iñigo, Montse Maritxalar, Aitor Gonzalez-Agirre, German Rigau, Larraitz Uria, and Eneko Agirre. "Interpretable semantic textual similarity: Finding and explaining differences between sentences." Knowledge-Based Systems 119 (2017): 186-199.

[9] D. Bollegala, Y. Matsuo, and M. Ishizuka, "Measuring semantic similarity between words using Web search engines," in Proc. WWW, vol. 7, 2007, pp. 757–766.

[10] D. Bollegala, Y. Matsuo, and M. Ishizuka, "Measuring semantic similarity between words using Web search engines," in Proc. WWW, vol. 7, 2007, pp. 757–766.

[11] R. L. Cilibrasi and P. M. B. Vitanyi, "The Google similarity distance," IEEE Trans. Knowl. Data Eng., vol. 19, no. 3, pp. 370–383, Mar. 2007.

[12] Atoum, Issa, and Ahmed Otoom. "Efficient hybrid semantic text similarity using WordNet and a corpus." Int. J. Adv. Comput. Sci. Appl 7, no. 9 (2016): 124-130.

[13] R. Ferreira, R. D. Lins, F. Freitas, S. J. Simske, and M. Riss, "A New Sentence Similarity Assessment Measure Based on a Three-layer Sentence Representation," in Proceedings of the 2014 ACM Symposium on Document Engineering, 2014, pp. 25–34

[14] Paetzold, Gustavo Henrique, and Lucia Specia. "Vicinity-driven paragraph and sentence alignment for comparable corpora." arXiv preprint arXiv:1612.04113 (2016).

Appendix

A. Importing the neccessary libraries

```
import os
from sklearn.utils import shuffle
import math
from nltk.tokenize import word_tokenize, sent_tokenize
from massalign.core import *
```

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
stop_words = set(stopwords.words('english'))
import string
import re
import datetime
import time
from numpy import array, argmax, random, take
import pandas as pd
from keras.models import Sequential
from sklearn.model_selection import train_test_split
from keras.layers import Dense, LSTM, Embedding, Bidirectional, RepeatVector, TimeDistributed
from keras.preprocessing.text import Tokenizer
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from keras import optimizers
import matplotlib.pyplot as plt
% matplotlib inline
pd.set_option('display.max_colwidth', 200)
from nltk.translate.bleu_score import sentence_bleu
```

B. Uploading the Newsela Extractor.py file

```
def files(path):
    for file in os.listdir(path):
        if os.path.isfile(os.path.join(path, file)):
            yield file
```

C. Formatting the raw Newsela data

```
#formatting Newsela raw data into format needed by aligner
rawDataPath = 'NewselaRaw'
formattedDataPath = 'NewselaFormatted'
ignoreFiles = []
for f in files(rawDataPath):
    print(f)
    if f in ignoreFiles:
        continue
    fileLines = [line.rstrip('\n') for line in open(rawDataPath+ '/' +f)]
    firstSent = fileLines[0]
```

```
    if firstSent.split(' ')[0].isupper():
        fileLines[0] = firstSent[firstSent.find("   ")+3:].strip()

    file_content = ''
    for l in fileLines:
        if l != '' and (not l.startswith("##")):
            file_content += l.strip() + ' '

    try:
        fileSentences = sent_tokenize(file_content)

        outFile = open(formattedDataPath + '/' + f, "w")
        for s in fileSentences:
            # write line to output file
            outFile.write(s)
            outFile.write("\n")
            outFile.write("\n")
        outFile.close()
    except:
        ignoreFiles.append(f)
```

D. Extracting the list of articles from the Newsela corpus

```
#get list of all articles in the corpus
articles = [line.rstrip('\n') for line in open('articleNames.csv')]
artDict = {}
for a in articles:
    tmpA = a.split(',')
    if tmpA[0] in artDict.keys():
        artDict[tmpA[0]].append(tmpA[5])
    else:
        artDict[tmpA[0]] = [tmpA[5]]
```

E. Generating the Normal and Simplified text files

```
path = 'alignedSentences-0.5.txt'
#output separated files for normal and simplified sentences
outSimpFile = open('alignedSentencesSimp-0.5.txt', "w")
outNormFile = open('alignedSentencesNorm-0.5.txt', "w")
lines = [line.rstrip('\n') for line in open(path)]
for l in lines:
    outNorm = l.split('\t')[0][3:-2] + "\n"
    outSimp = l.split('\t')[1][3:-2] + "\n"
    outNormFile.write(outNorm)
    outSimpFile.write(outSimp)
outNormFile.close()
outSimpFile.close()
```

F. For each of the sentence similarity percentage , combining the aligned sentence pairs

```
for sim in simVals:
    #for each similarity value from 0.3 to 0.7
    fileOutput = open("alignedSentences-" + str(sim) + ".txt","w")
    for k in artDict.keys():
        #combine data into aligned sentence pairs
        print k
        for i in range(len(artDict[k])-1):
            for j in range(i+1, len(artDict[k])):
                f1 = 'NewselaFormatted/' + artDict[k][i]
                f2 = 'NewselaFormatted/' + artDict[k][j]

                try:
                    aps = getAlignedParagraphs(f1, f2, sim)
                    for a in aps:
                        s = str(a[0]) + "\t" + str(a[1]) + "\n"
                        fileOutput.write(s)
                except:
                    continue
    fileOutput.close()
```

G. Opening the aligned sentence pairs text file and pre-processing the text files and building our model

```
def read_text(filename):
    # open the file
    file = open(filename, mode='rt', encoding='utf-8')
    # read all text
    text = file.read()
    file.close()
    return text

# split a text into sentences
def to_lines(text):
    sents = text.strip().split('\n')
    sents = [i.split('\t') for i in sents]
    return sents

# function to build a tokenizer
def tokenization(lines):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

# encode and pad sequences
def encode_sequences(tokenizer, length, lines):
    # integer encode sequences
    seq = tokenizer.texts_to_sequences(lines)
    # pad sequences with 0 values
    seq = pad_sequences(seq, maxlen=length, padding='post')
    return seq

# build NMT model
def build_model(in_vocab, out_vocab, in_timesteps, out_timesteps, hidden_size):
    model = Sequential()
```

```python
    model.add(Embedding(in_vocab, hidden_size, input_length=in_timesteps, mask_zero=True))
    model.add(LSTM(hidden_size))
    model.add(RepeatVector(out_timesteps))
    model.add(LSTM(hidden_size, return_sequences=True))
    model.add(Dense(out_vocab, activation='softmax'))
    return model

def get_word(n, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == n:
            return word
    return None

def calc_BLUE(pred_df):
    blue = 0
    for i in range(len(pred_df)):
        reference = [pred_df.iloc[i]['actual'].split()]
        candidate = pred_df.iloc[i]['predicted'].split()
        blue = blue + sentence_bleu(reference, candidate)
    return blue/len(pred_df)

def removeSentStopWords(sent):
  word_tokens = word_tokenize(sent.lower())
  filtered_sentence = [w for w in word_tokens if not w in stop_words]
  newSent = ''
  for s in range(len(filtered_sentence)):
    if s < len(filtered_sentence) - 1:
      if filtered_sentence[s+1].strip() in string.punctuation:
        newSent = newSent + filtered_sentence[s]
      else:
        newSent = newSent + filtered_sentence[s] + ' '
    else:
      newSent = newSent + filtered_sentence[s]
  return newSent
```

H. Building our auto-encoder model and initializing method to calculate bleu score

```python
def RunAutoEncoder(usingNewsela=True, word_cutoff = 15, aggr_length = 12, trainTestSplitPerc = 0.3,
                   hidden_size = 64, remove_punc = True, batchSize=64, remove_stopwords = False,
                   normFilePath='NewselaAlignedNorm-0.4.txt', simpFilePath='NewselaAlignedSimp-0.4.txt"'):
  if usingNewsela:
    data_normal = read_text(normFilePath)
    norm_data = to_lines(data_normal)
    norm_data = array(norm_data)
    norm_data = [s[0] for s in norm_data]

    data_simple = read_text(simpFilePath)
    simp_data = to_lines(data_simple)
    simp_data = array(simp_data)
    simp_data = [s[0] for s in simp_data]
  else:
    data_normal = read_text(normFilePath)
    norm_data = to_lines(data_normal)
    norm_data = array(norm_data)
    norm_data = [s[2] for s in norm_data]

    data_simple = read_text(simpFilePath)
    simp_data = to_lines(data_simple)
    simp_data = array(simp_data)
    simp_data = [s[2] for s in simp_data]


  #Remove stop words
  if remove_stopwords:
    print("Removving stop words...")
    for n in range(len(norm_data)):
      norm_data[n] = removeSentStopWords(norm_data[n])
    for s in range(len(simp_data)):
      simp_data[n] = removeSentStopWords(simp_data[n])
  allData= []
  allData_aggr=[]
  for i in range(len(norm_data)):
      if (len(norm_data[i].split()) < word_cutoff) and (len(simp_data[i].split()) < len(norm_data[i].split())):
          allData.append([norm_data[i], simp_data[i]])

  allData = array(allData)


  # Remove punctuation
  if remove_punc:
    print("Removing punctuations...")
    allData[:,0] = [s.translate(str.maketrans('', '', string.punctuation)) for s in allData[:,0]]
    allData[:,1] = [s.translate(str.maketrans('', '', string.punctuation)) for s in allData[:,1]]

  # convert to lowercase
  for i in range(len(allData)):
      allData[i,0] = allData[i,0].lower().strip()

      allData[i,1] = allData[i,1].lower().strip()
      allData_aggr.append(allData[i,0])
      allData_aggr.append(allData[i,1])

  # empty lists
  norm_l = []
  simp_l = []

```

```python
# populate the lists with sentence lengths
for i in allData[:,0]:
    norm_l.append(len(i.split()))

for i in allData[:,1]:
    simp_l.append(len(i.split()))

# prepare aggregate tokenizer
aggr_tokenizer = tokenization(allData_aggr)
aggr_vocab_size = len(aggr_tokenizer.word_index) + 1
# prepare normal tokenizer
norm_tokenizer = tokenization(allData[:, 0])
norm_vocab_size = len(norm_tokenizer.word_index) + 1
# prepare simple tokenizer
simp_tokenizer = tokenization(allData[:, 1])
simp_vocab_size = len(simp_tokenizer.word_index) + 1

train, test = train_test_split(allData, test_size=trainTestSplitPerc, random_state = 12)

# prepare training data
trainX = encode_sequences(aggr_tokenizer, aggr_length, train[:, 0])
trainY = encode_sequences(aggr_tokenizer, aggr_length, train[:, 1])

# prepare validation data
testX = encode_sequences(aggr_tokenizer, aggr_length, test[:, 0])
testY = encode_sequences(aggr_tokenizer, aggr_length, test[:, 1])

model = build_model(aggr_vocab_size, aggr_vocab_size, aggr_length, aggr_length, hidden_size)
rms = optimizers.RMSprop(lr=0.001)
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy')

filename = 'NLPFinalModel'
checkpoint = ModelCheckpoint(filename, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
start_time = time.time()
history = model.fit(trainX, trainY.reshape(trainY.shape[0], trainY.shape[1], 1),
          epochs=50, batch_size=batchSize,
          validation_split = 0.2,
          callbacks=[checkpoint], verbose=1)
print("--- %s seconds ---" % (time.time() - start_time))

model.summary()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['train','validation'])
plt.xlabel('# epochs')
plt.ylabel('Loss')
plt.title('Model training')
plt.show()

model = load_model('NLPFinalModel')
preds = model.predict_classes(testX.reshape((testX.shape[0],testX.shape[1])))

# convert predictions into text (English)
preds_text = []
for i in preds:
    temp = []
    for j in range(len(i)):
        t = get_word(i[j], aggr_tokenizer)
        if j > 0:
            if (t == get_word(i[j-1], aggr_tokenizer)) or (t == None):
                temp.append('')
            else:
                temp.append(t)

        else:
            if(t == None):
                temp.append('')
            else:
                temp.append(t)

    preds_text.append(' '.join(temp))

print(calc_BLUE(pd.DataFrame({'actual' : test[:,0], 'predicted' : preds_text})))
```

I. Calculating the bleu score for the normal and simplified text file

```python
#user variables
#simple wikipedia data
#normFilePath = "alignedSentencesNorm-0.3.txt"
#simpFilePath = "alignedSentencesSimp-0.3.txt"

#newsela corpus
normFilePath = "alignedSentencesNorm-0.3.txt"
simpFilePath = "alignedSentencesSimp-0.3.txt"

usingNewsela = True
remove_punc = True
remove_stopwords = False
word_cutoff = 15
aggr_length = 15
hidden_size = 256
batchSize = 64
trainTestSplitPerc= 0.3
RunAutoEncoder(usingNewsela, word_cutoff,
    aggr_length, trainTestSplitPerc,
    hidden_size, remove_punc, batchSize,
    remove_stopwords, normFilePath,
    simpFilePath)
```