National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Savin Vishwas Karkada
Student ID: x20184727

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Savin Vishwas Karkada
Student ID:	x20184727
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Christian Horn
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	14th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Savin Vishwas Karkada
x20184727

1 Introduction

The following manual provides details on hardware and software configuration which enables to reproduce the research work.

2 Hardware and Software Configuration

The following table provides an overview of the technologies, hardware settings and libraries with their description used in the research project.

Table 1: Setup Details.

Setup	**Description**
Computation	GPU
IDE	Jupyter Notebook, Google Colab
Programming Language	Python
Framework	Pytorch
Libraries	Sentence Transformer, Sklearn, Pandas, Numpy, Seaborn, ROUGE

The above tools and technologies were run on AMD Ryzen 4000 series with NVIDIA Geforce GTX to train, test and evaluate the models.

3 Dataset Details

The dataset is present inside the 'Data' folder which contains the data obtained from Hugging Face repository[1] CNN-Daily Mail. The subsets of data further used and created in the process of the research 'train_ data' and 'train5k' is also present in the same folder. The folder containing all the Jupyter notebooks also have the same set of CSV files placed for convenience. The website attached with Hugging Face points to the original repository where the entire dataset and the description of the dataset is mentioned. The dataset used here is directly downloaded from the link provided by Hugging Face.

[1]https://huggingface.co/datasets/cnn$_d$*ailymail*/*viewer*/3.0.0/*train*

4 Data Pre-Processing

The data Pre-Processing is carried out on the CSV file obtained from Hugging Face using Jupyter notebook. The iPYNB file 'Data Pre-Processing (EDA)' provides all the codes required to run EDA on the data. The Figure 1 shows the necessary libraries and packages that needs to be installed in prior to run the code seamlessly.

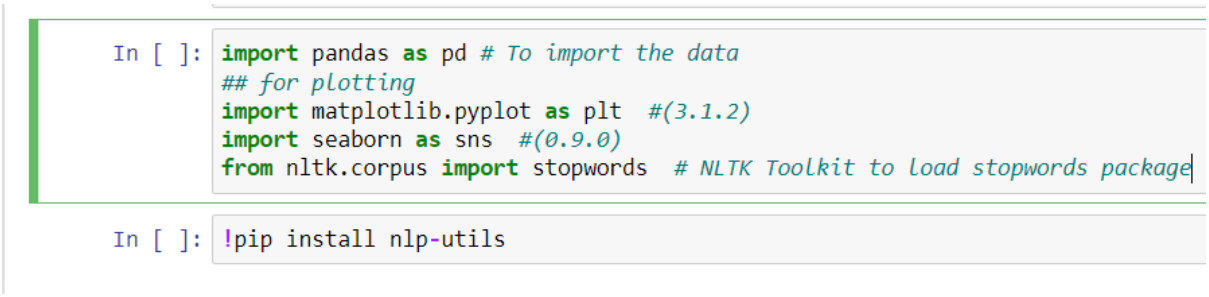

Figure 1: EDA Requirements

The following code snippet as represented in Figure 2 shows the method in which the data is loaded into Jupyter notebook and to display the data. Here the dataset 'train.csv' is used which contains the entire data from Hugging Face CNN-Daily Mail dataset.

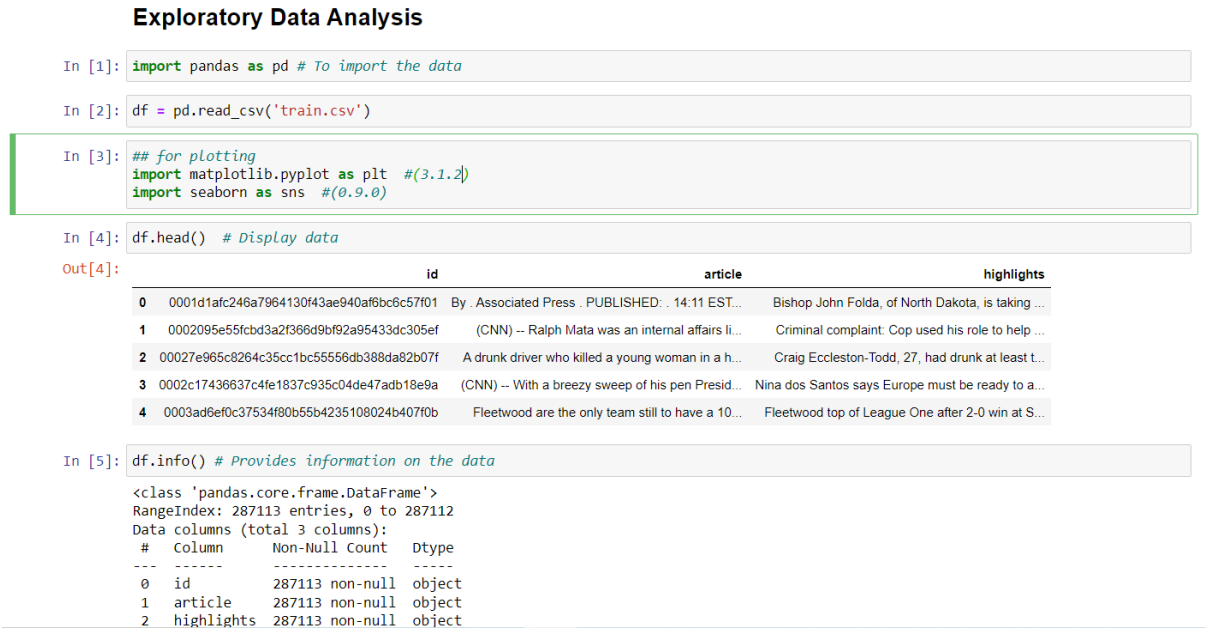

Figure 2: Data Load and Display

The Figure 3 shows the packages and tools derived from NLP toolkit to eliminate stop words and contractions. This data is further used to train various models in the research. The pre-processing such as analysing the sentence length, understanding the summary of the dataset and reducing the size of the dataset to a subset to carry out further operations are all done in this phase.Once the pre-processed data is ready the data is converted into a seperate CSV file which is then used in training and summarizing different models in different notebooks. Allt he required packages are provided in the file requirements.txt present in the 'Models' folder.

Installing NLP text cleaning requirements (stopwords & contractions)

```
In [7]: !pip install nlp-utils

Requirement already satisfied: nlp-utils in d:\anaconda1\lib\site-packages (0.6.2)
Requirement already satisfied: micro-toolkit in d:\anaconda1\lib\site-packages (from nlp-utils) (0.9.0)
Requirement already satisfied: nltk in d:\anaconda1\lib\site-packages (from nlp-utils) (3.7)
Requirement already satisfied: numpy in d:\anaconda1\lib\site-packages (from nlp-utils) (1.21.5)
Requirement already satisfied: tqdm in d:\anaconda1\lib\site-packages (from nltk->nlp-utils) (4.64.0)
Requirement already satisfied: click in d:\anaconda1\lib\site-packages (from nltk->nlp-utils) (8.0.4)
Requirement already satisfied: regex>=2021.8.3 in d:\anaconda1\lib\site-packages (from nltk->nlp-utils) (2022.3.15)
Requirement already satisfied: joblib in d:\anaconda1\lib\site-packages (from nltk->nlp-utils) (1.1.0)
Requirement already satisfied: colorama in d:\anaconda1\lib\site-packages (from click->nltk->nlp-utils) (0.4.4)

In [8]: from nltk.corpus import stopwords  # NLTK Toolkit to load stopwords package
        stop = stopwords.words('english')
```

Figure 3: Pre-Processing

5 Model Implementation

5.1 Word2Vec

Word2Vec is the first embedder that is implemented using Jupyter notebook in the model implementation phase. The requirements for installation and imports are shown in the Figure 4 Here, the embedding process is done directly using the test data as there is

```
In [ ]: import nltk
        nltk.download('punkt')   # one time execution
        from nltk.tokenize import sent_tokenize
        import re                        # Importing RegEX
        nltk.download('stopwords')  # one time execution
        from nltk.corpus import stopwords
        from gensim.models import Word2Vec
        import numpy as np
        from sklearn.cluster import KMeans
        from scipy.spatial import distance
        import rouge  #(1.0.0)
        from rouge import Rouge

In [ ]: !pip install rouge
        !pip install rouge-score
```

Figure 4: Word2Vec Requirements

no training required for Word2Vec embedder. The Figure 5 shows the method to load Word2Vec model from the Gensim library.

From the loaded Gensim library, the model Word2Vec is derived which then creates embeddings when the test article is provided. The test article is given in the 'Data' folder in txt file called 'Test Article'. The embeddings from Word2Vec is fed into the K-Means clustering model to generate the summary. The method to implement K-Means clustering from SKlearn is shown in the Figure 6. The necessary installation to load the model is provided in the requirements.txt folder provided in the 'Models' folder.

Word2Vec Embedding Using Gensim

```
In [ ]: from gensim.models import Word2Vec
        all_words = [i.split() for i in corpus]
        model = Word2Vec(all_words, min_count=1,size= 300)
```

```
In [ ]: sent_vector=[]
        for i in corpus:

            plus=0
            for j in i.split():
                plus+= model.wv[j]
            plus = plus/len(i.split())

            sent_vector.append(plus)
```

Figure 5: Word2Vec Embedding

K-Means Clustering

```
In [ ]: import numpy as np
        from sklearn.cluster import KMeans
        n_clusters = 5
        kmeans = KMeans(n_clusters, init = 'k-means++', random_state = 42)
        y_kmeans = kmeans.fit_predict(sent_vector)
```

```
In [ ]: from scipy.spatial import distance
        my_list=[]
        for i in range(n_clusters):
            my_dict={}

            for j in range(len(y_kmeans)):

                if y_kmeans[j]==i:
                    my_dict[j] =  distance.euclidean(kmeans.cluster_centers_[i],sent_vector[j])
            min_distance = min(my_dict.values())
            my_list.append(min(my_dict, key=my_dict.get))


        for i in sorted(my_list):
            print(sentence[i])
```

```
Two of them are in serious condition, the company said.
Authorities evacuated about 300 people from the Abkatun Permanente platform after the fire started, Pemex said.
At least 10 boats worked to battle the blaze for hours.
The state oil company hasn't said what caused the fire on the platform, which is located in the Gulf of Mexico's Campeche Soun
d.
CNN's Mayra Cuevas contributed to this report.
```

Figure 6: K-Means Clustering

Finally, the evaluation of the model is done using ROUGE scores and the ROUGE requirements are shown in the Figure 4 and provided in requirements.txt.

5.2 ELMo Embedding

ELMo contextual embedder is implemented using Google Colab[2] as the version of ELMo - ELMo2 does not support the Tensorflow 2 version. Inorder to stepdown the TF version Google Colab would be an ideal IDE as it can be easily changed without altering the environment settings. Hence using the code shown in Figure 7 can lower the version to effectively load and use ELMo model. The figure also points out all the necessary package imports that are essential to load and run all the packages to seamlessly run the embedding and summarization process. The evaluation packages used are provided in the requirements.txt file which can be also used in a Google Colab environment. In this implementation, we do not train the ELMo model and hence no data is imported and only test article is defined to feed the embedder.

[2]https://colab.research.google.com/?utm$_source = scs - index$

```
In [ ]: %tensorflow_version 1.x
        !python --version
```

```
In [ ]: import tensorflow_hub as hub
        import tensorflow as tf
        import re
        import tensorflow as tf
        import tensorflow_hub as hub
        import numpy as np
        import pandas as pd
        import time
        import string
        from spacy.lang.en import English # updated
        from sklearn.cluster import KMeans
        from sklearn.metrics import pairwise_distances_argmin_min
        import rouge  #(1.0.0)
        from rouge import Rouge
```

```
In [ ]: !pip install rouge
        !pip install rouge-score
```

Figure 7: ELMo Requirements

ELMo Embedding Phase

```
In [ ]: def create_embedding(x):   # Load ELMo model
          #tf hub module
          elmo = hub.Module("https://tfhub.dev/google/elmo/2", trainable=True)
          embeddings = elmo(x, signature="default", as_dict=True)["elmo"]
```

```
In [ ]: def create_embedding(x):
          #tf hub module
          elmo = hub.Module("https://tfhub.dev/google/elmo/2", trainable=True)
          embeddings = elmo(x, signature="default", as_dict=True)["elmo"]

          # embeddings.shape
          with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
            sess.run(tf.tables_initializer())
            # return average of ELMo features
            return sess.run(tf.reduce_mean(embeddings,1))

        def lemmatization(texts):
          output = []
          for i in nlp(texts):
            output.append(i.lemma_)
          output = " ".join(i for i in output)
          return output
```

Figure 8: ELMo Embedding

The ELMo 2 is fetched from the ELMo repository[3]. The implementation of ELMo to create embeddings is as shown in the Figure 8. The clustering and evaluation process is similar to the previous embedders used and the required installation packages are provided in requirements.txt.

5.3 BERT Embedding

BERT embeddings are obtained by loading a pre-trained BERT model from SBERT[4] repository. The initial requirements to load the BERT model and import packages are as shown in Figure 9.

As the pre-trained model here will be trained on the dataset to fine tune the model the data that is pre-processed is loaded. The data is by the name 'train5k'. The initialized

[3]https://tfhub.dev/google/elmo/2
[4]https://www.sbert.net/

```
In [ ]: import nltk                  # Importing NLTK for basic pre-processing
nltk.download('punkt')
from sentence_transformers import SentenceTransformer, LoggingHandler
from sentence_transformers import models, util, datasets, evaluation, losses
from torch.utils.data import DataLoader
import re
import pandas as pd
from nltk.cluster import KMeansClusterer
import numpy as np
from sentence_transformers import SentenceTransformer
from scipy.spatial import distance_matrix

In [ ]: !pip install -U sentence-transformers  # Install Sentence Transformer
!pip install rouge
!pip install rouge-score
```

Figure 9: BERT Requirements

BERT model to train is called 'bert-base-uncased'. The fine tuning is unsupervised and is achieved by the use of TSADE model that is provided by SBERT for the fine tuning purposes. The details to load the model and train is as shown in Figure 10

BERT Training Phase (Fine-tuning)

```
In [2]: from sentence_transformers import SentenceTransformer, LoggingHandler
from sentence_transformers import models, util, datasets, evaluation, losses
from torch.utils.data import DataLoader

In [3]: batch_size = 128
# The following approaches only require sentences from your target domain.

# Define your sentence transformer model using CLS pooling
model_name = 'bert-base-uncased'

In [4]: word_embedding_model = models.Transformer(model_name)
pooling_model = models.Pooling(word_embedding_model.get_word_embedding_dimension(), 'cls')
model_S = SentenceTransformer(modules=[word_embedding_model, pooling_model])

Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertModel: ['cls.predictions.decoder.
weight', 'cls.predictions.transform.dense.bias', 'cls.seq_relationship.weight', 'cls.predictions.bias', 'cls.predictions.transf
orm.LayerNorm.bias', 'cls.seq_relationship.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.dens
e.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another arc
hitecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
```

Data Loading and Cleanup

```
In [5]: import pandas as pd

In [6]: df = pd.read_csv('train5k')
```

Figure 10: BERT Initializing

The Figure 11 shows the training of BERT model after which the model generates BERT embeddings which is further fed into K-Means clustering algorithm in a similar fashion as shown in the previous models. Here the K-Means clustering model is obtained from the SKlearn repository. The parameters for the clustering algorithm remains the same as that of the previous models. The required packages are provided in requirements.txt file. The model that is trained and clustered provides summary of size 5 which then is evaluated using ROUGE scores in a similar manner as executed in the previous embedders.

Training TSDAE BERT Model

```
In [19]: model_S.fit(
             train_objectives=[(train_dataloader, train_loss)],
             epochs=100,
             weight_decay=0.01,
             scheduler='constantlr',
             optimizer_params={'lr': 3e-5},
             show_progress_bar=True
         )

         model_S.save('output/tsdae-model')

         Epoch:   0%|          | 0/100 [00:00<?, ?it/s]
         Iteration:   0%|          | 0/1 [00:00<?, ?it/s]
         Iteration:   0%|          | 0/1 [00:00<?, ?it/s]
         Iteration:   0%|          | 0/1 [00:00<?, ?it/s]
         Iteration:   0%|          | 0/1 [00:00<?, ?it/s]
         Iteration:   0%|          | 0/1 [00:00<?, ?it/s]
         Iteration:   0%|          | 0/1 [00:00<?, ?it/s]
         Iteration:   0%|          | 0/1 [00:00<?, ?it/s]
         Iteration:   0%|          | 0/1 [00:00<?, ?it/s]
         Iteration:   0%|          | 0/1 [00:00<?, ?it/s]
         Iteration:   0%|          | 0/1 [00:00<?, ?it/s]
```

Figure 11: BERT Training

5.4 RoBERTa Embedding

The process of initializing, training and embedding of RoBERTa is same as that of BERT which is achieved from the SBERT repository. The requirements to load and run RoBERTa from SBERT is as shown in the Figure 12.

```
In [ ]: import nltk                      # Importing NLTK for basic pre-processing
        nltk.download('punkt')
        from sentence_transformers import SentenceTransformer, LoggingHandler
        from sentence_transformers import models, util, datasets, evaluation, losses
        from torch.utils.data import DataLoader
        import re
        import pandas as pd
        from nltk.cluster import KMeansClusterer
        import numpy as np
        from sentence_transformers import SentenceTransformer
        from scipy.spatial import distance_matrix

In [ ]: !pip install -U sentence-transformers  # Install Sentence Transformer
        !pip install rouge
        !pip install rouge-score
```

Figure 12: RoBERTa Requirements

The model used to train RoBERTa is a pre-trained by the name 'roberta-base' from the TSADE repository. The training details and parameters are as shown below in the Figure 13. The training set used is the 'train5k' data CSV obtained from the same folder. Finally the model is trained on the given dataset.

The model after training is fed with the test article which undergoes RoBERTa embedding. These embeddings are further fed into K-Means clusters to generate summary as done in the previous embeddings. The results are similarly evaluated using ROUGE scores and the requirements are given in the requirements.txt file.

RoBERTa Training Phase (Fine-tuning)

In [3]:
```
from sentence_transformers import SentenceTransformer, LoggingHandler
from sentence_transformers import models, util, datasets, evaluation, losses
from torch.utils.data import DataLoader
```

In [4]:
```
batch_size = 128
# The following approaches only require sentences from your target domain.

# Define your sentence transformer model using CLS pooling
model_name = 'roberta-base'
```

In [5]:
```
word_embedding_model = models.Transformer(model_name)
pooling_model = models.Pooling(word_embedding_model.get_word_embedding_dimension(), 'cls')
model_S = SentenceTransformer(modules=[word_embedding_model, pooling_model])
```

```
Some weights of the model checkpoint at roberta-base were not used when initializing RobertaModel: ['lm_head.bias', 'lm_head.de
nse.bias', 'lm_head.layer_norm.bias', 'lm_head.decoder.weight', 'lm_head.dense.weight', 'lm_head.layer_norm.weight']
- This IS expected if you are initializing RobertaModel from the checkpoint of a model trained on another task or with another
architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing RobertaModel from the checkpoint of a model that you expect to be exactly identi
cal (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
```

Data Loading and Cleanup

In [6]:
```
import pandas as pd
```

In [7]:
```
df = pd.read_csv('train5k')
```

Figure 13: RoBERTa Initializing