Configuration Manual

MSc Research Project
Data Analytics

Mathiazhagan Sampath
Student ID: x18139973

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Mathiazhagan Sampath
Student ID:	x18139973
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Muhammad Iqbal
Submission Due Date:	12/12/2019
Project Title:	Configuration Manual
Word Count:	580
Page Count:	14

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:	11th December 2019

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

Mathiazhagan Sampath
x18139973

1 Introduction

This configuration Manual is to provide information on System Configuration, Programming Language and Program code used to implement the research project:
"Toxic Question Classification in Question & Answer forum using Deep Learning"

2 System Configuration

Kaggle Kernal is used to build the proposed model. Kaggle kernel are 12.5x faster on training Deep learning models.

2.1 Hardware

1. OS: Linux
2. Hard Disk: 5GB
3. RAM: 16 GB
4. GPU Enabled.
5. Processor: Intel Core i7

2.2 Software

1. Kaggle Notebook - Python
   (a) Data Extraction
   (b) Explanatory Data Analysis (EDA)
   (c) Visualization
   (d) Feature Extraction
   (e) Data Cleaning
   (f) Building the Deep Learning Model
2. Microsoft Excel
   (a) Pivot Table
   (b) Visualization

3 Project Development

Development of Classification model to classify question in Q&A forum is of many phases. Starting from Data Extraction, Explanatory Data Analysis, Data Cleaning, Embedding matrix creation from Glove and fastText, Sequence creation of texts in questions. Recurrent Neural Network model will be built using CuDNNLSTM and CuDNNGRU to build the final neural network model. In this section data extraction, EDA will be done.

3.1 Data Collection

1. Data downloaded from Quora
2. Explanatory Data Analysis
3. Glove Embedding Layer data from Stanford
4. fastText Embedding layer from Google.

Embedding layer are zipped and uploaded to Kaggle kernel.

3.2 Data Preparation

Data required to build the models are

1. Quora Question & Answer dataset
2. Embedding Layer for Transfer learning from Stanford and Google Repository

Two kernels are created in kaggle for this research purpose. Click on the kernel and follow the steps in image to fork and run the kernel to see the output.

1. Explanatory Data Analysis of dataset `https://www.kaggle.com/mathiazhagan/toxic-classification-kernal/` Figure 1 represent the steps to fork the kernel of EDA

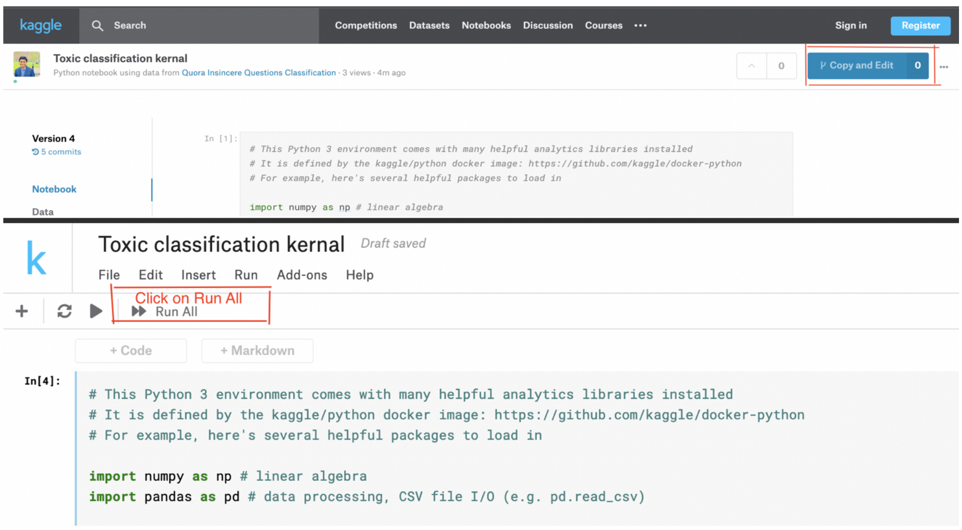

Figure 1: Kaggle EDA Kernel Fork

2. NLP and Model Development
   https://www.kaggle.com/mathiazhagan/cycliclr-and-k-fold. Figure 2 represent the steps to fork the kernel of Model development.

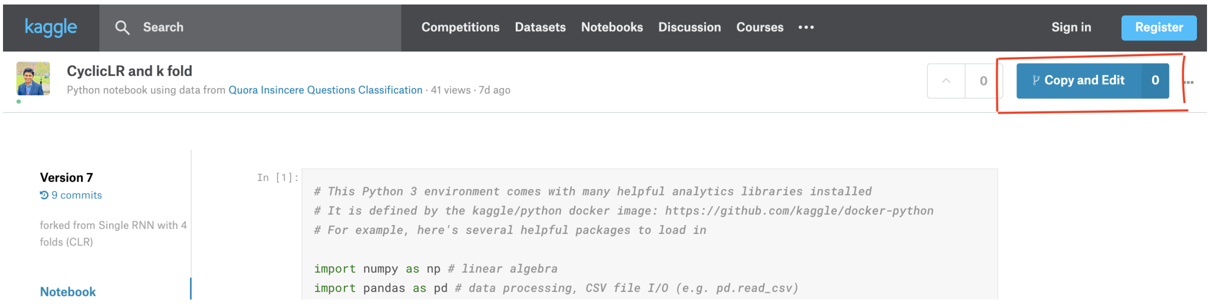

Figure 2: Kaggle Model developed Kernel Fork

4 Explanatory Data Analysis

Navigate to [1] to fork kernel. Detailed EDA of the dataset will be done in this section. Figure 3 provide the information on libraries imported.

```
df_train=pd.read_csv("/kaggle/input/quora-insincere-questions-classification/train.csv")
df_test=pd.read_csv("/kaggle/input/quora-insincere-questions-classification/test.csv")
```

Import libraries

```
[36]:
import os
import re
import csv
import string
import gc
from tqdm import tqdm

import numpy as np
import pandas as pd
import seaborn as sns
from collections import Counter
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
from scipy.sparse import hstack
from IPython.display import Image
from prettytable import PrettyTable

from tqdm import tqdm_notebook
tqdm_notebook().pandas()

from nltk.stem import PorterStemmer, SnowballStemmer, WordNetLemmatizer
from nltk.stem.lancaster import LancasterStemmer
from nltk.util import ngrams
```

Figure 3: Import of Required Libraries

[1] https://www.kaggle.com/mathiazhagan/toxic-classification-kernal/

Figure 4, 5 represent the Class imbalance in the dataset.

EDA of the Quora Questions

Class Imbalance of toxic and non-toxic

+ Code + Markdown

In[37]:

```
values = [df_train[df_train['target']==0].shape[0], df_train[df_train['target']==1].shape[0]]
labels = ['Non Toxic questions', 'Toxic questions']

plt.pie(values, labels=labels, autopct='%1.1f%%', shadow=True)
plt.title('Target Distribution')
plt.tight_layout()
plt.subplots_adjust(right=1.9)
plt.show()
```

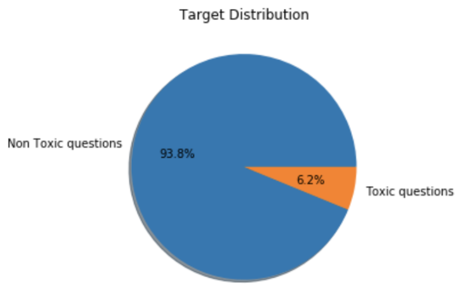

Figure 4: Class Imbalance in data

Bar graph of class Imbalance

```
cnt_srs = df_train['target'].value_counts()
trace = go.Bar(
    x=cnt_srs.index,
    y=cnt_srs.values,
    marker=dict(
        color=cnt_srs.values,
        colorscale = 'Picnic',
        reversescale = True
    ),
)

layout = go.Layout(
    title='Target Count',
    font=dict(size=18)
)

data = [trace]
fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename="TargetCount")
```

Figure 5: Bar graph of class Imbalance

4.1 Feature Extraction from Questions

Figure 6, 7, 8 provides the step involved in feature extraction using Python, Visualisation of features is done on BoxPlot and Correlation Matrix is drawn on the same using Python plotly.

Feature Extraction of Question from dataset

```
# Number of words
df_train['num_words'] = df_train['question_text'].apply(lambda x: len(str(x).split()))
df_test['num_words'] = df_test['question_text'].apply(lambda x: len(str(x).split()))

# Number of capital_letters
df_train['num_capital_let'] = df_train['question_text'].apply(lambda x: len([c for c in str(x) if c.isupper()]))
df_test['num_capital_let'] = df_test['question_text'].apply(lambda x: len([c for c in str(x) if c.isupper()]))

# Number of special characters
df_train['num_special_char'] = df_train['question_text'].str.findall(r'[^a-zA-Z0-9 ]').str.len()
df_test['num_special_char'] = df_test['question_text'].str.findall(r'[^a-zA-Z0-9 ]').str.len()

# Number of unique words
df_train['num_unique_words'] = df_train['question_text'].apply(lambda x: len(set(str(x).split())))
df_test['num_unique_words'] = df_test['question_text'].apply(lambda x: len(set(str(x).split())))

# Number of numerics
df_train['num_numerics'] = df_train['question_text'].apply(lambda x: sum(c.isdigit() for c in x))
df_test['num_numerics'] = df_test['question_text'].apply(lambda x: sum(c.isdigit() for c in x))

# Number of characters
df_train['num_char'] = df_train['question_text'].apply(lambda x: len(str(x)))
df_test['num_char'] = df_test['question_text'].apply(lambda x: len(str(x)))

# Number of stopwords
df_train['num_stopwords'] = df_train['question_text'].apply(lambda x: len([c for c in str(x).lower().split() if c in STOPWORDS]))
df_test['num_stopwords'] = df_test['question_text'].apply(lambda x: len([c for c in str(x).lower().split() if c in STOPWORDS]))
```

Figure 6: Feature Extraction of data

```
def display_boxplot(_x, _y, _data, _title):
    sns.boxplot(x=_x, y=_y, data=_data)
    plt.grid(True)
    #plt.tick_params(axis='x', which='major', labelsize=15)
    plt.title(_title,fontsize=20)
    plt.xlabel(_x, fontsize=15)

# Boxplot: Number of words
plt.subplot(2, 3, 1)
display_boxplot('target', 'num_words', df_train, 'No. of words in each class')

# Boxplot: Number of chars
plt.subplot(2, 3, 2)
display_boxplot('target', 'num_char', df_train, 'Number of characters in each class')

# Boxplot: Number of unique words
plt.subplot(2, 3, 3)
display_boxplot('target', 'num_unique_words', df_train, 'Number of unique words in each class')

# Boxplot: Number of special characters
plt.subplot(2, 3, 4)
display_boxplot('target', 'num_special_char', df_train, 'No. of special characters in each class')

# Boxplot: Number of stopwords
plt.subplot(2, 3, 5)
display_boxplot('target', 'num_stopwords', df_train, 'Number of stopwords in each class')

# Boxplot: Number of capital letters
plt.subplot(2, 3, 6)
display_boxplot('target', 'num_capital_let', df_train, 'No. of capital letters in each class')


plt.subplots_adjust(right=3.0)
plt.subplots_adjust(top=2.0)
plt.show()
```

Figure 7: Box Plot of the Feature Extraction

In[]:
```
# Correlation matrix
f, ax = plt.subplots(figsize=(10, 8))
corr = df_train.corr()
sns.heatmap(corr, ax=ax,annot=True)
plt.title("Correlation matrix")
plt.show()
```

Figure 8: Correlation Matrix of Extracted Feature

4.2 NGRAM visualization of Questions

Most commonly or frequently used unigram(1), Bigram(2), Trigram (3) are extracted and visualized. Figure 9, 10 provide the detail steps involved in the process

```
from collections import defaultdict
from nltk.corpus import stopwords
from nltk import WordNetLemmatizer
from plotly import tools
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
stop_words = set(stopwords.words('english'))
insinc_df = df_train[df_train.target==1]
sinc_df = df_train[df_train.target==0]
def plot_ngrams(n_grams):
    ## custom function for ngram generation ##
    def generate_ngrams(text, n_gram=1):
        token = [token for token in text.lower().split(" ") if token != "" if token not in stop_words]
        ngrams = zip(*[token[i:] for i in range(n_gram)])
        return [" ".join(ngram) for ngram in ngrams]
    ## custom function for horizontal bar chart ##
    def horizontal_bar_chart(df, color):
        trace = go.Bar(
            y=df["word"].values[::-1],
            x=df["wordcount"].values[::-1],
            showlegend=False,
            orientation = 'h',
            marker=dict(
                color=color,
            ),
        )
        return trace
    def get_bar(df, bar_color):
        freq_dict = defaultdict(int)
        for sent in df["question_text"]:
            for word in generate_ngrams(sent, n_grams):
                freq_dict[word] += 1
        fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x[1])[::-1])
        fd_sorted.columns = ["word", "wordcount"]
```

Figure 9: Ngram Visualisation (Contd).

```
                freq_dict[word] += 1
        fd_sorted = pd.DataFrame(sorted(freq_dict.items(), key=lambda x: x[1])[::-1])
        fd_sorted.columns = ["word", "wordcount"]
        trace = horizontal_bar_chart(fd_sorted.head(13), bar_color)
        return trace
    trace0 = get_bar(sinc_df, 'green')
    trace1 = get_bar(insinc_df, 'red')
    # Creating two subplots
    if n_grams == 1:
        wrd = "words"
    elif n_grams == 2:
        wrd = "Bigrams"
    elif n_grams == 3:
        wrd = "Trigrams"
    fig = toolsmake_subplots(rows=1, cols=2, vertical_spacing=0.03,subplot_titles=["Frequent " + wrd + " of Toxic",
            "Frequent " + wrd + " of Non-toxic "])
    fig.append_trace(trace0, 1, 1)
    fig.append_trace(trace1, 1, 2)
    fig['layout'].update(height=500, width=750, paper_bgcolor='rgb(233,233,233)', title=wrd + " Count Plots")
    py.iplot(fig, filename='word-plots')
```

+ Code + Markdown

```
#Unigram
plot_ngrams(1)
#Bigram
plot_ngrams(2)
#Trigram
plot_ngrams(3)
```

Figure 10: Ngram Visualization.

5 Making the class balance

Navigate to [2] to access the model developed in Kaggle. Figure 11 represent the configuration feature and class balance implementation.

```
embed_size = 300 # how big is each word vector
max_features = 61000 # how many unique words to use (i.e num rows in embedding vector)
maxlen = 70 # max number of words in a question to use
```

```
train=pd.read_csv("../input/train.csv")
# Class count
count_class_0, count_class_1 = train.target.value_counts()

# Divide by class
df_class_0 = train[train['target'] == 0]
df_class_1 = train[train['target'] == 1]
df_class_0_under = df_class_0.sample(count_class_1)
train = pd.concat([df_class_0_under, df_class_1], axis=0)

print('Random under-sampling:')
print(train.target.value_counts())
```

```
Random under-sampling:
1    80810
0    80810
Name: target, dtype: int64
```

+ Code + Markdown

```
train['question_text']
```

Figure 11: Configuration and Class Balance

[2] https://www.kaggle.com/mathiazhagan/cycliclr-and-k-fold

6 Cleaning of Data

Figure 12, 13, 14, 15 represent the data cleaning process involved in cleaning the dataset after sampling.

```python
from collections import defaultdict
from nltk.corpus import stopwords
from nltk import WordNetLemmatizer
import re
def get_symbols_in_one(data):
    symbols=[]
    for text in data:
        s=re.findall(r'\W+',str(text))
        symbols.extend(s),
    return set(symbols)
#Symbols presnet in the dataset
final_symbols=get_symbols_in_one(train["question_text"].values)

def clean_tag(x):
    if '[math]' in x:
        x=re.sub('\[math\].*?math\]','MATH EQUATION',x)
    if 'http' in x or 'www' in x:
        x=re.sub('(?:(?:https?|ftp):\/\/)?[\w/\-?=%.]+\.[\w/\-?=%.]+', 'URL', x)
    return x
def clean_tag(x):
    if '[math]' in x:
        x=re.sub('\[math\].*?math\]','MATH EQUATION',x)
    if 'http' in x or 'www' in x:
        x=re.sub('(?:(?:https?|ftp):\/\/)?[\w/\-?=%.]+\.[\w/\-?=%.]+', 'URL', x)
    return x
```

Figure 12: Data Cleaning of tags in questions

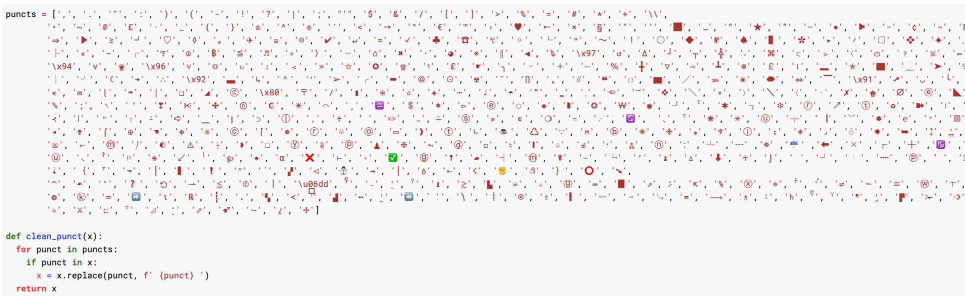

Figure 13: Data cleaning of symbols in question

```python
mispell_dict = {'colour': 'color', 'centre': 'center', 'favourite': 'favorite', 'travelling': 'traveling', 'counselling': 'counseling', 'theatre': 'theater',
                'cancelled': 'canceled', 'labour': 'labor', 'organisation': 'organization',
                'wwii': 'world war 2', 'citicise': 'criticize', 'youtu ': 'youtube ', 'Qoura': 'Quora',
                'sallary': 'salary', 'Whta': 'What', 'narcisist': 'narcissist', 'howdo': 'how do', 'whatare': 'what are', 'howcan': 'how can',
                'howmuch': 'how much', 'howmany': 'how many', 'whydo': 'why do', 'doI': 'do I', 'theBest': 'the best', 'howdoes': 'how does',
                'mastrubation': 'masturbation', 'mastrubate': 'masturbate', "mastrubating": 'masturbating', 'pennis': 'penis',
                'Etherium': 'bitcoin', 'narcissit': 'narcissist', 'bigdata': 'big data', '2k17': '2017', '2k18': '2018', 'qouta': 'quota', 'exboyfriend': 'ex boyfriend',
                'airhostess': 'air hostess', "whst": 'what', 'watsapp': 'whatsapp',
                'demonitisation': 'demonetization', 'demonitization': 'demonetization', 'demonetisation': 'demonetization',
                'electroneum':'bitcoin','nanodegree':'degree','hotstar':'star','dream11':'dream','ftre':'fire','tensorflow':'framework','unocoin':'bitcoin',
                'lnmiit':'limit','unacademy':'academy','altcoin':'bitcoin','altcoins':'bitcoin','litecoin':'bitcoin','coinbase':'bitcoin','cryptocurency':'cryptocurrency',
                'simpliv':'simple','quoras':'quora','schizoids':'psychopath','remainers':'remainder','twinflame':'soulmate','quorans':'quora','brexit':'demonetized',
                'iiest':'institute','dceu':'comics','pessat':'exam','uceed':'college','bhakts':'devotee','boruto':'anime',
                'cryptocoin':'bitcoin','blockchains':'blockchain','fiancee':'fiance','redmi':'smartphone','oneplus':'smartphone','qoura':'quora','deepmind':'framework',
                'ryzen':'cpu','whattsapp':'whatsapp',
                'undertale':'adventure','zenfone':'smartphone','cryptocurencies':'cryptocurrencies','koinex':'bitcoin','zebpay':'bitcoin',
                'binance':'bitcoin','whtsapp':'whatsapp',
                'reactjs':'framework','bittrex':'bitcoin','bitconnect':'bitcoin','bitfinex':'bitcoin',
                'yourquote':'your quote','whyis':'why is','jiophone':'smartphone',
                'dogecoin':'bitcoin','onecoin':'bitcoin','poloniex':'bitcoin','7700k':'cpu','angular2':'framework',
                'segwit2x':'bitcoin','hashflare':'bitcoin','940mx':'gpu',
                'openai':'framework','hashflare':'bitcoin','1050ti':'gpu','nearbuy':'near buy','freebitco':'bitcoin',
                'antminer':'bitcoin','filecoin':'bitcoin','whatapp':'whatsapp',
                'empowr':'empower','1080ti':'gpu','crytocurrency':'cryptocurrency','8700k':'cpu','whatsaap':'whatsapp','g4560':'cpu','payymoney':'pay money',
                'fuckboys':'fuck boys','intenship':'internship','zcash':'bitcoin','demonatisation':'demonetization','narcicist':'narcissist','mastuburation':'masturbation',
                'trignometric':'trigonometric','cryptocurreny':'cryptocurrency','howdid':'how did','crytocurrencies':'cryptocurrencies','phycopath':'psychopath',
                'bytecoin':'bitcoin','possesiveness':'possessiveness','scollege':'college','humanties':'humanities','altacoin':'bitcoin','demonitised':'demonetized',
                'brasília':'brazilia','accolite':'accolyte','econimics':'economics','varrier':'warrier','quroa':'quora','statergy':'strategy','langague':'language',
                'splatoon':'game','7600k':'cpu','gate2018':'gate 2018','in2018':'in 2018','narcassist':'narcissist','jiocoin':'bitcoin','hnlu':'hulu','7300hq':'cpu',
                'weatern':'western','interledger':'blockchain','deplation':'deflation', 'cryptocurrencies':'cryptocurrency', 'bitcoin':'blockchain cryptocurrency',}
def correction_mispell(x):
    word=x.split()
    for i in range(0,len(word)):
        if mispell_dict.get(word[i]) is not None:
            word[i]=mispell_dict.get(word[i])
        elif mispell_dict.get(word[i].lower()) is not None:
            word[i]=mispell_dict.get(word[i].lower)
        word=" ".join(word)
        return word
```

Figure 14: Data cleaning of Misspelled words in Questions

```
def clear_other_contradiction(x):
    special=["’", "‘", "´", "`"]
    for s in special:
        x=x.replace(s,"'")
    x = ' '.join([contraction_mapping[n] if n in contraction_mapping else n for n in x.split(" ")])
    return x
lemmatizer = WordNetLemmatizer()
def lemma_text(x):
  x = x.split()
  x = [lemmatizer.lemmatize(word) for word in x]
  x = ' '.join(x)

  return x
def data_cleaning(x):
    clean_tag(x)
    clean_punct(x)
    correction_mispell(x)
    clear_other_contradiction(x)
    lemma_text(x)
    return x
train['question_text']=train['question_text'].apply(lambda x:data_cleaning(x))
```

Figure 15: Implementation of data cleaning

7 Model Development

7.1 Split of data

Figure 16 represent the implementation of the data split into train,test for the model.

```
embed_size = 300 # how big is each word vector
max_features = 61000 # how many unique words to use (i.e num rows in embedding vector)
maxlen = 70 # max number of words in a question to use
def load_and_prec():
    print("Entered")
    train_df,test_df, = train_test_split(train,test_size=0.20, random_state=2018)

    ## fill up the missing values
    train_X = train_df["question_text"].fillna("_##_").values
    test_X = test_df["question_text"].fillna("_##_").values

    ## Tokenize the sentences
    tokenizer = Tokenizer(num_words=max_features)
    tokenizer.fit_on_texts(list(train_X))
    train_X = tokenizer.texts_to_sequences(train_X)
    test_X = tokenizer.texts_to_sequences(test_X)

    ## Pad the sentences
    train_X = pad_sequences(train_X, maxlen=maxlen)
    test_X = pad_sequences(test_X, maxlen=maxlen)

    ## Get the target values
    train_y = train_df['target'].values
    test_y=test_df['target'].values

    #shuffling the data
    np.random.seed(2018)
    trn_idx = np.random.permutation(len(train_X))

    train_X = train_X[trn_idx]
    train_y = train_y[trn_idx]
    return train_X, test_X, train_y, test_y,tokenizer.word_index
train_X, test_X, train_y,test_y, word_index = load_and_prec()
```

Figure 16: Load and split the data

7.2 Load of Transfer Learning Layers

Figure 17 represent the Transfer learning techniques involved in creating embedding matrix.

```python
def load_glove(word_index):
    EMBEDDING_FILE = '../input/embeddings/glove.840B.300d/glove.840B.300d.txt'
    def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')
    embeddings_index = dict(get_coefs(*o.split(" ")) for o in open(EMBEDDING_FILE))

    all_embs = np.stack(embeddings_index.values())
    emb_mean,emb_std = all_embs.mean(), all_embs.std()
    embed_size = all_embs.shape[1]

    # word_index = tokenizer.word_index
    nb_words = min(max_features, len(word_index))
    embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))
    for word, i in word_index.items():
        if i >= max_features: continue
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None: embedding_matrix[i] = embedding_vector
    return embedding_matrix

def load_fasttext(word_index):
    EMBEDDING_FILE = '../input/embeddings/wiki-news-300d-1M/wiki-news-300d-1M.vec'
    def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')
    embeddings_index = dict(get_coefs(*o.split(" ")) for o in open(EMBEDDING_FILE) if len(o)>100)

    all_embs = np.stack(embeddings_index.values())
    emb_mean,emb_std = all_embs.mean(), all_embs.std()
    embed_size = all_embs.shape[1]

    # word_index = tokenizer.word_index
    nb_words = min(max_features, len(word_index))
    embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words, embed_size))
    for word, i in word_index.items():
        if i >= max_features: continue
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None: embedding_matrix[i] = embedding_vector
    return embedding_matrix
```

Figure 17: Load of Embedding layer - Transfer Learning

7.3 Attention Layer

Figure 18 represent the implementation of Attention layer for getting providing higher weights for the important words in the questions.

```python
#https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html
#https://arxiv.org/abs/1706.03762
#https://arxiv.org/abs/1409.0473

class Attention(Layer):
    def __init__(self, step_dim,
                 W_regularizer=None, b_regularizer=None,
                 W_constraint=None, b_constraint=None,
                 bias=True, **kwargs):
        self.supports_masking = True
        self.init = initializers.get('glorot_uniform')
        self.W_regularizer = regularizers.get(W_regularizer)
        self.b_regularizer = regularizers.get(b_regularizer)
        self.W_constraint = constraints.get(W_constraint)
        self.b_constraint = constraints.get(b_constraint)
        self.bias = bias
        self.step_dim = step_dim
        self.features_dim = 0
        super(Attention, self).__init__(**kwargs)
    def build(self, input_shape):
        assert len(input_shape) == 3

        self.W = self.add_weight((input_shape[-1],),initializer=self.init,name='{}_W'.format(self.name),regularizer=self.W_regularizer,constraint=self.W_constraint)
        self.features_dim = input_shape[-1]
        if self.bias:
            self.b = self.add_weight((input_shape[1],),initializer='zero',name='{}_b'.format(self.name),regularizer=self.b_regularizer,constraint=self.b_constraint)
        else:
            self.b = None
        self.built = True
    def compute_mask(self, input, input_mask=None):
        return None
    def call(self, x, mask=None):
        features_dim = self.features_dim
        step_dim = self.step_dim
        eij = K.reshape(K.dot(K.reshape(x, (-1, features_dim)),
                        K.reshape(self.W, (features_dim, 1))), (-1, step_dim))
        if self.bias:
            eij += self.b
        eij = K.tanh(eij)
        a = K.exp(eij)
        if mask is not None:
            a *= K.cast(mask, K.floatx())
        a /= K.cast(K.sum(a, axis=1, keepdims=True) + K.epsilon(), K.floatx())
        a = K.expand_dims(a)
        weighted_input = x * a
        return K.sum(weighted_input, axis=1)
    def compute_output_shape(self, input_shape):
        return input_shape[0],  self.features_dim
```

Figure 18: Implementation of Attention Class

7.4 Cyclic Learning Rate(CLR)

Figure 19, 20 represent the implementation of predicting the learning rate for training the Neural Network.

1. Different Learning Rate from 0.001 to 0.002
2. Different Learning Rate from 0.001 to 0.003
3. Different Learning Rate from 0.001 to 0.004
4. gamma=0.9994
5. mode=exp_range

```
#CLR
class CyclicLR(Callback):
    def __init__(self, base_lr=0.001, max_lr=0.006, step_size=2000., mode='triangular',
                 gamma=1., scale_fn=None, scale_mode='cycle'):
        super(CyclicLR, self).__init__()

        self.base_lr = base_lr
        self.max_lr = max_lr
        self.step_size = step_size
        self.mode = mode
        self.gamma = gamma
        if scale_fn == None:
            if self.mode == 'triangular':
                self.scale_fn = lambda x: 1.
                self.scale_mode = 'cycle'
            elif self.mode == 'triangular2':
                self.scale_fn = lambda x: 1/(2.**(x-1))
                self.scale_mode = 'cycle'
            elif self.mode == 'exp_range':
                self.scale_fn = lambda x: gamma**(x)
                self.scale_mode = 'iterations'
        else:
            self.scale_fn = scale_fn
            self.scale_mode = scale_mode
        self.clr_iterations = 0.
        self.trn_iterations = 0.
        self.history = {}

        self._reset()

    def _reset(self, new_base_lr=None, new_max_lr=None,
               new_step_size=None):
        """Resets cycle iterations.
        Optional boundary/step size adjustment.
        """
        if new_base_lr != None:
            self.base_lr = new_base_lr
        if new_max_lr != None:
            self.max_lr = new_max_lr
        if new_step_size != None:
            self.step_size = new_step_size
```

Figure 19: Implementation of Cyclic Learning Rate (Contd).

```
        self.clr_iterations = 0.

    def clr(self):
        cycle = np.floor(1+self.clr_iterations/(2*self.step_size))
        x = np.abs(self.clr_iterations/self.step_size - 2*cycle + 1)
        if self.scale_mode == 'cycle':
            return self.base_lr + (self.max_lr-self.base_lr)*np.maximum(0, (1-x))*self.scale_fn(cycle)
        else:
            return self.base_lr + (self.max_lr-self.base_lr)*np.maximum(0, (1-x))*self.scale_fn(self.clr_iterations)

    def on_train_begin(self, logs={}):
        logs = logs or {}

        if self.clr_iterations == 0:
            K.set_value(self.model.optimizer.lr, self.base_lr)
        else:
            K.set_value(self.model.optimizer.lr, self.clr())

    def on_batch_end(self, epoch, logs=None):

        logs = logs or {}
        self.trn_iterations += 1
        self.clr_iterations += 1

        self.history.setdefault('lr', []).append(K.get_value(self.model.optimizer.lr))
        self.history.setdefault('iterations', []).append(self.trn_iterations)

        for k, v in logs.items():
            self.history.setdefault(k, []).append(v)

        K.set_value(self.model.optimizer.lr, self.clr())
```

Figure 20: Implementation of Cyclic Learning Rate

7.5 F1 score method for the model

Figure 21 represent the implementation of method to find the F1 score based on the precision and Recall value of the model.

```
def f1(y_true, y_pred):
    '''
    metric from here
    https://stackoverflow.com/questions/43547402/how-to-calculate-f1-macro-in-keras
    '''
    def recall(y_true, y_pred):
        """Recall metric.

        Only computes a batch-wise average of recall.

        Computes the recall, a metric for multi-label classification of
        how many relevant items are selected.
        """
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

    def precision(y_true, y_pred):
        """Precision metric.

        Only computes a batch-wise average of precision.

        Computes the precision, a metric for multi-label classification of
        how many selected items are relevant.
        """
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision
    precision = precision(y_true, y_pred)
    recall = recall(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Figure 21: F1 score calculation using Precision and Recall

7.6 Creation of Embedded Matrix

Figure 22 represent the Creation of the Embedding Matrix using Glove and fastText.

```
In[]:
embedding_matrix_glove = load_glove(word_index)
embedding_matrix_fast = load_fasttext(word_index)
embedding_matrix = np.mean([embedding_matrix_glove,embedding_matrix_fast], axis = 0)
np.shape(embedding_matrix)
## Simple average: http://aclweb.org/anthology/N18-2031
## Unweighted DME in https://arxiv.org/pdf/1804.07983.pdf
embedding_matrix = np.mean([embedding_matrix_glove,embedding_matrix_fast], axis = 0)
np.shape(embedding_matrix)
```

+ Code   + Markdown

```
def threshold_search(y_true, y_proba):
    best_threshold = 0
    best_score = 0
    for threshold in np.arange(0.1, 0.501, 0.01):
    #for threshold in [i * 0.01 for i in range(100)]:
        threshold = np.round(threshold, 2)
        #precision=precision_score(y_true=y_true, y_pred=y_proba > threshold)
        #recall=recall_score(y_true=y_true, y_pred=y_proba > threshold)
        score = f1_score(y_true=y_true, y_pred=y_proba > threshold)
        print("F1 score at threshold {0} is {1} ".format(threshold,score))
        if score > best_score:
            best_threshold = threshold
            best_score = score
            recall=recall_score
            precision=precision_score
    search_result = {'threshold': best_threshold, 'f1': best_score}
    print("Best thershold {0} and F1 score is {1} ".format(best_threshold,best_score))
    return search_result
```

Figure 22: Creation of Embedding matrix from Glove and fastText and Threshold search method

7.7 Final Model with different number of Layer and LR

Figure 23 represent the split on train,test and Validation data, 24 represent the model implementation with different Learning Rate, number of nodes in hidden layers.

```
def train_pred(model, train_X, train_y, val_X, val_y, epochs=2, callback=None):
    for e in range(epochs):
        model.fit(train_X, train_y, batch_size=512, epochs=1, validation_data=(val_X, val_y), callbacks = callback, verbose=0)
        pred_val_y = model.predict([val_X], batch_size=1024, verbose=0)
        best_score = metrics.f1_score(val_y, (pred_val_y >0.38).astype(int))
        print("Epoch: ", e, "-    Val F1 Score: {:.4f} ".format(best_score))
    pred_test_y = model.predict([test_X], batch_size=1024, verbose=0)
    print('=' * 60)
    return pred_val_y, pred_test_y, best_score
```

```
def model_lstm_atten1(embedding_matrix):
    inp = Input(shape=(maxlen,))
    x = Embedding(max_features, embed_size, weights=[embedding_matrix], trainable=False)(inp)
    x = SpatialDropout1D(0.1)(x)
    x = Bidirectional(CuDNNLSTM(256, return_sequences=True))(x)
    y = Bidirectional(CuDNNGRU(128, return_sequences=True))(x)

    atten_1 = Attention(maxlen)(x) # skip connect
    atten_2 = Attention(maxlen)(y)
    avg_pool = GlobalMaxPooling1D()(y)
    max_pool = GlobalMaxPooling1D()(y)

    conc = concatenate([atten_1, atten_2, avg_pool, max_pool])
    outp = Dense(1, activation="sigmoid")(conc)
    model = Model(inputs=inp, outputs=outp)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=[f1])

    return model
```

Figure 23: Train and test of dataset on the Model with High number of nodes in Hidden layer

```
#With lower number of node and other parameters
def model_lstm_atten(embedding_matrix):
    inp = Input(shape=(maxlen,))
    x = Embedding(max_features, embed_size, weights=[embedding_matrix], trainable=False)(inp)
    x = SpatialDropout1D(0.1)(x)
    x = Bidirectional(CuDNNLSTM(40, return_sequences=True))(x)
    y = Bidirectional(CuDNNGRU(40, return_sequences=True))(x)
    atten_1 = Attention(maxlen)(x) # skip connect
    atten_2 = Attention(maxlen)(y)
    avg_pool = GlobalAveragePooling1D()(y)
    max_pool = GlobalMaxPooling1D()(y)
    conc = concatenate([atten_1, atten_2, avg_pool, max_pool])
    conc = Dense(16, activation="relu")(conc)
    conc = Dropout(0.1)(conc)
    outp = Dense(1, activation="sigmoid")(conc)
    model = Model(inputs=inp, outputs=outp)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=[f1])
    return model
```

Figure 24: Implentation of Model 2 with less number of node in Hidden Layer.

8 Code Referencec

1. Smith (2015)
2. Coates and Bollegala (2018)
3. Kiela et al. (2018)
4. Vaswani et al. (2017)
5. Keras github[3]
6. Keras Discussion Community[4]
7. Stackoverflow[5]

References

Coates, J. and Bollegala, D. (2018). Frustratingly easy meta-embedding – computing meta-embeddings by averaging source word embeddings, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, Association for Computational Linguistics, New Orleans, Louisiana, pp. 194–198.
URL: *https://www.aclweb.org/anthology/N18-2031*

Kiela, D., Wang, C. and Cho, K. (2018). Dynamic meta-embeddings for improved sentence representations, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Brussels, Belgium, pp. 1466–1477.
URL: *https://www.aclweb.org/anthology/D18-1176*

Smith, L. N. (2015). Cyclical learning rates for training neural networks.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I. (2017). Attention is all you need, *CoRR* **abs/1706.03762**.
URL: *http://arxiv.org/abs/1706.03762*

[3] `https://github.com/keras-team/keras/wiki/Keras-2.0-release-notes`
[4] `https://github.com/keras-team/keras/issues/5794#issuecomment-287641301`
[5] `https://stackoverflow.com/questions/43547402/how-to-calculate-f1-macro-in-keras`