National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Ashish Patel
Student ID: x18182445

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland
MSc Project Submission Sheet
School of Computing

National College of Ireland

Student Name: Ashish Patel……………………………………………………………………………………

Student ID: X18182445……………………………………………………………………………………………

Programme: Data Analytics………………………………………… **Year:** 2019-2020

Module: MSc Research Project…………………………………………………………………………

Supervisor: Christian Horn………………………………………………………………………………

Submission Due Date: 17/08/2020……………………………………………………………………………….

Project Title: Understanding Subjective Aspect of Question Answer …………………

Word Count 1200………….. **Page Count** 18……………………………………..

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: 17/08/2020……………………………………………………………………………………………

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

Ashish Patel
x18182445

1 Introduction

This manual gives a proper description and the steps taken while performing the research to check if Deep Learning architectures are capable of recognizing the subjective aspect of the question and answer. Below are the configuration of the system on which the research was carried on:

- Intel Core I5 Processor with 8 GB RAM and x64 processor

2 Requirements

The complete research was done using the Google Colab notebook, for running the project in the local machine python 3.7 and TensorFlow 2.0 are required and some pre-trained model which can be downloaded from below links.

- BERT model : https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/2
- Universal Sentence Encoder Model: https://tfhub.dev/google/universal-sentence-encoder/4
- GloVe Pre trained Embedding can be downloaded from the link: https://nlp.stanford.edu/projects/glove/

3 Dataset

The dataset used in the research were downloaded as csv file from the Kaggle website which hosted the Google competition, through the link https://www.kaggle.com/c/google-quest-challenge/data

4 Exploratory Data Analysis and Data Cleaning

First the data was downloaded from the Kaggle website then all the important libraries are imported for the project

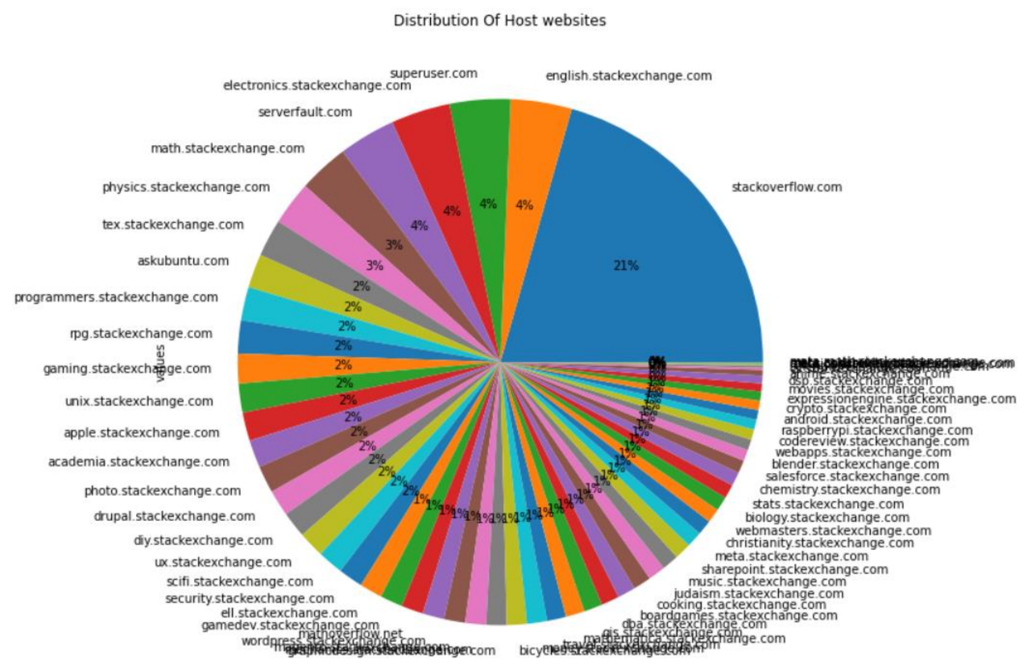
```
# import os
import sys
import glob
import torch
import re
import gc
import random

from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
```

```python
import numpy as np
import pandas as pd
import math
import os
import sys
#import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import re
import random
from urllib.parse import urlparse
from math import floor, ceil
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()
import plotly.offline as py
py.init_notebook_mode(connected=True)
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.offline as offline
offline.init_notebook_mode()
#import cufflinks and offline mode
import cufflinks as cf
cf.go_offline()

#import plotly.figure_factory as ff
#from plotly.subplots import make_subplots
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

from tqdm.notebook import tqdm
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import KFold, GroupKFold
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from scipy.stats import spearmanr, rankdata
```

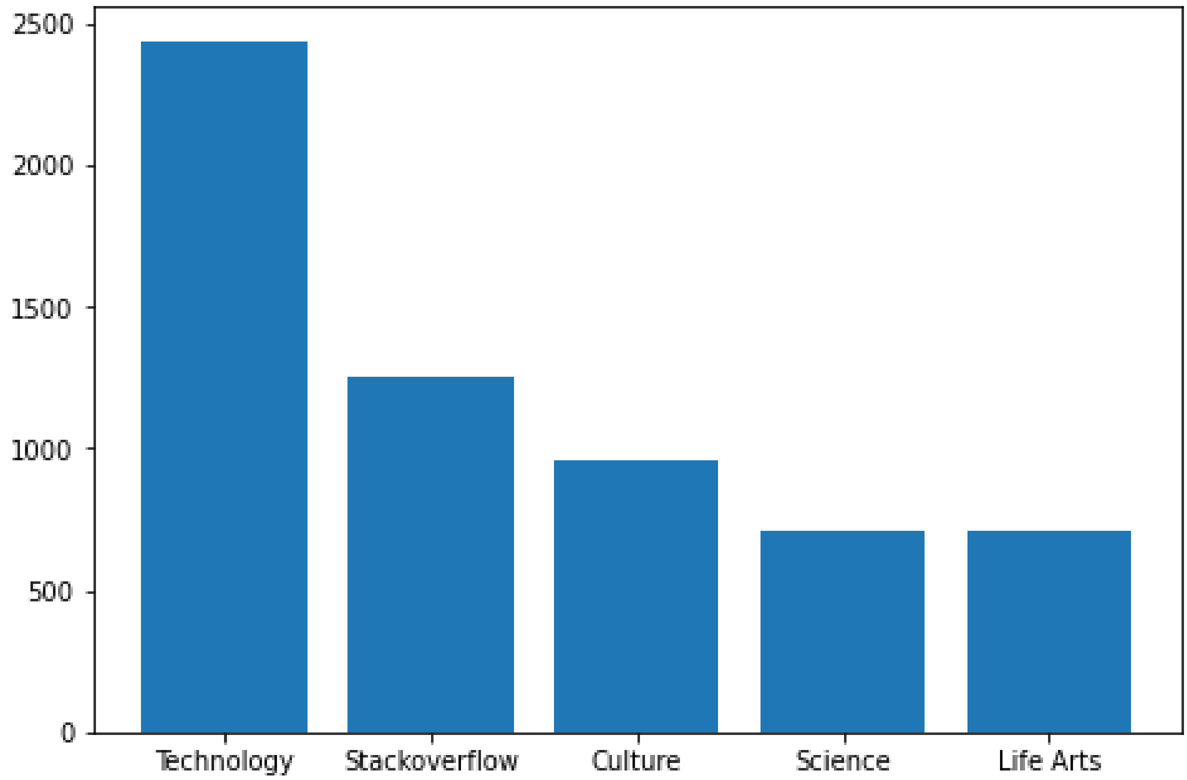
```python
import tensorflow as tf
import tensorflow_hub as hub
from keras.preprocessing import text, sequence
from keras.models import Model
from keras.layers import Input, LSTM, Bidirectional, Conv1D, Dense, Dropout, GlobalAveragePooling1D, Embedding, Activation, Lambd
from keras.optimizers import Adam, RMSprop
from keras.callbacks import Callback, ReduceLROnPlateau
import keras.backend as K

from sklearn.linear_model import MultiTaskElasticNet

sys.path.insert(0, '/content/drive/My Drive/ThesisProject')
import tfv2_tokenization
```

Trian.csv file is then loaded into DataFrame and some preliminary analysis of data is done to understand the data.

```python
temp = train_df["host"].value_counts()
df = pd.DataFrame({'labels': temp.index,
                   'values': temp.values
                  })
df = df.set_index('labels')
plot = df.plot.pie(y='values', autopct='%1.0f%%',figsize=(10, 10))
plot.set_label(False)
plot.get_xaxis().set_visible(False)
plot.set_title('Distribution Of Host websites')
plot.get_legend().remove()
```




```
temp = train_df["category"].value_counts()
print(temp)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
cat= ['Technology','Stackoverflow','Culture','Science','Life Arts']
value = temp
ax.bar(cat,value)
plot.set_title('Total number of question answer pairs from different categories')
plt.show()
```



Ongoing through the above bar chart we could make out that most question answer pair were technological category and from pie chart we could make out the most question were from StackOverflow website. Moving ahead with cleaning of the text data.

```
mispell_dict = {
    "aren't" : "are not","can't" : "cannot","couldn't" : "could not","couldnt" : "could not","didn't" : "did not",
"doesn't" : "does not","doesnt" : "does not","don't" : "do not","here's" : "here is","hadn't" : "had not","hasn't" : "has not",
"haven't" : "have not","havent" : "have not","he'd" : "he would","he'll" : "he will","he's" : "he is","i'd" : "I would","i'll" :
"i'm" : "I am","isn't" : "is not","it's" : "it is","it'll":"it will","i've" : "I have","let's" : "let us","mightn't" : "might not
"mustn't" : "must not","shan't" : "shall not","she'd" : "she would","she'll" : "she will","she's" : "she is","shouldn't" : "shoul
"shouldnt" : "should not","that's" : "that is","thats" : "that is","there's" : "there is","theres" : "there is","they'd" : "they
"they'll" : "they will","they're" : "they are","theyre":  "they are","they've" : "they have","we'd" : "we would","we're" : "we ar
"weren't" : "were not","we've" : "we have","what'll" : "what will","what're" : "what are","what's" : "what is","what've" : "what
"who'd" : "who would","who'll" : "who will","who're" : "who are","who's" : "who is","who've" : "who have","won't" : "will not",
"wouldn't" : "would not","you'd" : "you would","you'll" : "you will","you're" : "you are","you've" : "you have",
"'re": " are","wasn't": "was not","we'll":" will","didn't": "did not","tryin'":"trying"
}
```

```
regexp_tok = RegexpTokenizer('[A-Za-z]+')

def clean_text(x):

    def replace(match):
        return mispell_dict[match.group(0)]

    x_low = x.lower()
    misspell_re = re.compile('(%s)' % '|'.join(mispell_dict.keys()))
    correct_x = misspell_re.sub(replace, x_low)
    x = ' '.join(regexp_tok.tokenize(correct_x))
    return x

train_df['question_title'] = train_df['question_title'].apply(lambda x: clean_text(x))
train_df['question_body'] = train_df['question_body'].apply(lambda x: clean_text(x))
train_df['answer'] = train_df['answer'].apply(lambda x: clean_text(x))

#test_df['question_title'] = test_df['question_title'].apply(lambda x: clean_text(x))
#test_df['question_body'] = test_df['question_body'].apply(lambda x: clean_text(x))
#test_df['answer'] = test_df['answer'].apply(lambda x: clean_text(x))
```

Text data is cleaned with help of regular expressions and misspell dictionary, moving ahead with the data pre-processing and modelling of BERT model.

5 Creating a custom callback function

For the evaluation part while training, a custom callback function was used, which at the end of every epoch will calculate the spearman correlation of all the 30 validation attributes with the predicted attributes and give the average results, this custom callback function is used while training of all the three models.

```
[ ] class SpearmanRhoCallback(Callback):
        def __init__(self, training_data, validation_data, patience, model_name=None):
            self.x = training_data[0]
            self.y = training_data[1]
            self.x_val = validation_data[0]
            self.y_val = validation_data[1]

            self.patience = patience
            self.value = -1
            self.bad_epochs = 0
            self.model_name = model_name

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

        def on_train_end(self, logs={}):
            return

        def on_epoch_begin(self, epoch, logs={}):
            return

        def on_epoch_end(self, epoch, logs={}):
            y_pred_val = self.model.predict(self.x_val)
            if np.count_nonzero(self.y_val[:, 19]) == 0:
                self.y_val[:, 19] = self.y_val[:, 19] + np.random.normal(0, 1e-7, self.y_val.shape[0])

            rho_val = np.mean([spearmanr(self.y_val[:, ind], y_pred_val[:, ind] + np.random.normal(0, 1e-7, y_pred_val.shape[0])).correlation
            if rho_val >= self.value:
                self.value = rho_val
            else:
                self.bad_epochs += 1
            if self.bad_epochs >= self.patience:
```

```
        if self.bad_epochs >= self.patience:
            print("Epoch %05d: early stopping Threshold" % epoch)
            self.model.stop_training = True
            #self.model.save_weights(self.model_name)
        print('\rval_spearman-rho: %s' % (str(round(rho_val, 4))), end=100*' '+'\n')
        return rho_val

    def on_batch_begin(self, batch, logs={}):
        return

    def on_batch_end(self, batch, logs={}):
        return
```

While calculating the results or score for the comparison of the model, average correlation score of all 30 attributes in the test dataset with the predicted attributes is done, and the average spearman correlation score is the final score for all the three models.

6 Data Pre-Processing for BERT Model Training

BERT model takes input data in a specific format with [CLS], [SEP] and [MASK] tokens below code converts the text data into numerical data as expected by the BERT model

```
def _get_masks(tokens, max_seq_length):
    """Mask for padding"""
    if len(tokens)>max_seq_length:
        raise IndexError("Token length more than max seq length!")
    return [1]*len(tokens) + [0] * (max_seq_length - len(tokens))

def _get_segments(tokens, max_seq_length):
    """Segments: 0 for the first sequence, 1 for the second"""
    if len(tokens)>max_seq_length:
        raise IndexError("Token length more than max seq length!")
    segments = []
    first_sep = True
    current_segment_id = 0
    for token in tokens:
        segments.append(current_segment_id)
        if token == "[SEP]":
            if first_sep:
                first_sep = False
            else:
                current_segment_id = 1
    return segments + [0] * (max_seq_length - len(tokens))

def _get_ids(tokens, tokenizer, max_seq_length):
    """Token ids from Tokenizer vocab"""
    token_ids = tokenizer.convert_tokens_to_ids(tokens)
    input_ids = token_ids + [0] * (max_seq_length-len(token_ids))
    return input_ids
```

```python
def _trim_input(title, question, answer, max_sequence_length,
                t_max_len=30, q_max_len=239, a_max_len=239):

    t = tokenizer.tokenize(title)
    q = tokenizer.tokenize(question)
    a = tokenizer.tokenize(answer)

    t_len = len(t)
    q_len = len(q)
    a_len = len(a)

    if (t_len+q_len+a_len+4) > max_sequence_length:

        if t_max_len > t_len:
            t_new_len = t_len
            a_max_len = a_max_len + floor((t_max_len - t_len)/2)
            q_max_len = q_max_len + ceil((t_max_len - t_len)/2)
        else:
            t_new_len = t_max_len

        if a_max_len > a_len:
            a_new_len = a_len
            q_new_len = q_max_len + (a_max_len - a_len)
        elif q_max_len > q_len:
            a_new_len = a_max_len + (q_max_len - q_len)
            q_new_len = q_len
        else:
            a_new_len = a_max_len
            q_new_len = q_max_len
```

```python
        if t_new_len+a_new_len+q_new_len+4 != max_sequence_length:
            raise ValueError("New sequence length should be %d, but is %d"
                             % (max_sequence_length, (t_new_len+a_new_len+q_new_len+4)))

        t = t[:t_new_len]
        q = q[:q_new_len]
        a = a[:a_new_len]

    return t, q, a

def _convert_to_bert_inputs(title, question, answer, tokenizer, max_sequence_length):
    """Converts tokenized input to ids, masks and segments for BERT"""

    stoken = ["[CLS]"] + title + ["[SEP]"] + question + ["[SEP]"] + answer + ["[SEP]"]

    input_ids = _get_ids(stoken, tokenizer, max_sequence_length)
    input_masks = _get_masks(stoken, max_sequence_length)
    input_segments = _get_segments(stoken, max_sequence_length)

    return [input_ids, input_masks, input_segments]

def compute_input_arays(df, columns, tokenizer, max_sequence_length):
    input_ids, input_masks, input_segments = [], [], []
    for _, instance in tqdm(df[columns].iterrows()):
        t, q, a = instance.question_title, instance.question_body, instance.answer

        t, q, a = _trim_input(t, q, a, max_sequence_length)

        ids, masks, segments = _convert_to_bert_inputs(t, q, a, tokenizer, max_sequence_length)
        input_ids.append(ids)
```

```
        input_masks.append(masks)
        input_segments.append(segments)

    return [np.asarray(input_ids, dtype=np.int32),
            np.asarray(input_masks, dtype=np.int32),
            np.asarray(input_segments, dtype=np.int32)]
```

```
[ ] #tokenized data
    train_inputs = compute_input_arays(train_df, ['question_title', 'question_body', 'answer'], tokenizer, MAX_SEQUENCE_LENGTH)
    #test_inputs = compute_input_arays(test_df, ['question_title', 'question_body', 'answer'], tokenizer, MAX_SEQUENCE_LENGTH)
```

```
6079/? [21:27<00:00, 4.72it/s]
```

From the above output we could make out that 6079 rows of text data got converted to the BERT input format.

7 BERT Model Training

Taking the formatted input and then building a BERT model with extra global average pooling 1D layer to make the output in one dimesion and then having another dense layer with sigmoid activation function as an output

.

```
def bert_model():

    input_word_ids = tf.keras.layers.Input(
        (MAX_SEQUENCE_LENGTH,), dtype=tf.int32, name='input_word_ids')
    input_masks = tf.keras.layers.Input(
        (MAX_SEQUENCE_LENGTH,), dtype=tf.int32, name='input_masks')
    input_segments = tf.keras.layers.Input(
        (MAX_SEQUENCE_LENGTH,), dtype=tf.int32, name='input_segments')

    bert_layer = hub.KerasLayer(BERT_PATH, trainable=True)

    _, sequence_output = bert_layer([input_word_ids, input_masks, input_segments])

    x = tf.keras.layers.GlobalAveragePooling1D()(sequence_output)
    #x = tf.keras.layers.Concatenate()([x, input_ohe])
    #x = tf.keras.layers.Dropout(0.2)(x)
    #x = tf.keras.layers.Dense(256, activation='elu')(x)
    x = tf.keras.layers.Dropout(0.2)(x)
    out = tf.keras.layers.Dense(30, activation="sigmoid", name="dense_output")(x)

    model = tf.keras.models.Model(
        inputs=[input_word_ids, input_masks, input_segments], outputs=out)

    return model
```

```
model = bert_model()
model.summary()
```

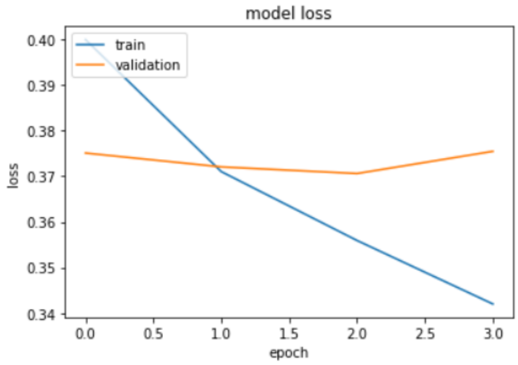
```
Model: "functional_5"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_word_ids (InputLayer)     [(None, 512)]        0
__________________________________________________________________________________________________
input_masks (InputLayer)        [(None, 512)]        0
__________________________________________________________________________________________________
input_segments (InputLayer)     [(None, 512)]        0
__________________________________________________________________________________________________
keras_layer_2 (KerasLayer)      [(None, 768), (None, 109482241   input_word_ids[0][0]
                                                                 input_masks[0][0]
                                                                 input_segments[0][0]
__________________________________________________________________________________________________
global_average_pooling1d_2 (Glo (None, 768)          0           keras_layer_2[0][1]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 768)          0           global_average_pooling1d_2[0][0]
__________________________________________________________________________________________________
dense_output (Dense)            (None, 30)           23070       dropout_2[0][0]
==================================================================================================
Total params: 109,505,311
Trainable params: 109,505,310
Non-trainable params: 1
__________________________________________________________________________________________________
```

Once the model is defined now the model is trained directly and with K-fold cross validation but the training results seems better in the K-fold cross validation technique as shown below.

```
all_predictions = []
gkf = GroupKFold(n_splits=5).split(X=train_df.question_body, groups=train_df.question_body)

for fold, (tr_idx, val_idx) in enumerate(gkf):

    K.clear_session()

    X_tr = [train_inputs[i][tr_idx] for i in range(3)]
    y_tr = y_train[tr_idx]
    X_val = [train_inputs[i][val_idx] for i in range(3)]
    y_val = y_train[val_idx]

    model = bert_model()
    model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=0.00005))
    history=model.fit(X_tr, y_tr,
                      epochs=4,
                      batch_size=8,
                      validation_data=(X_val, y_val),
                      callbacks=[SpearmanRhoCallback(training_data=(X_tr, y_tr),
                                                     validation_data=(X_val, y_val), patience=5)])
    all_predictions.append(model.predict(test_inputs))

    if fold==2: break
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
Epoch 1/4
val_spearman-rho: 0.3572
608/608 [==============================] - 761s 1s/step - loss: 0.4020 - val_loss: 0.3758
Epoch 2/4
val_spearman-rho: 0.372
608/608 [==============================] - 761s 1s/step - loss: 0.3725 - val_loss: 0.3706
Epoch 3/4
val_spearman-rho: 0.3894
608/608 [==============================] - 761s 1s/step - loss: 0.3588 - val_loss: 0.3664
Epoch 4/4
val_spearman-rho: 0.3785
608/608 [==============================] - 762s 1s/step - loss: 0.3459 - val_loss: 0.3735
Epoch 1/4
val_spearman-rho: 0.3466
608/608 [==============================] - 764s 1s/step - loss: 0.3986 - val_loss: 0.3792
Epoch 2/4
val_spearman-rho: 0.3664
608/608 [==============================] - 762s 1s/step - loss: 0.3693 - val_loss: 0.3765
val_spearman-rho: 0.385
608/608 [==============================] - 761s 1s/step - loss: 0.3559 - val_loss: 0.3706
Epoch 4/4
val_spearman-rho: 0.3782
608/608 [==============================] - 763s 1s/step - loss: 0.3420 - val_loss: 0.3755
```



Once the model is trained we could see that the spearman correlation score is 0.37 for the validation data when we run the test data on the model we could get the score of 0.35

```
all_predictions.append(model.predict(xtest))
```

```
all_predictions
```

```
[array([[9.1985255e-01, 6.7840552e-01, 7.8855287e-03, ..., 2.9397009e-02,
         6.4647323e-01, 9.3610549e-01],
        [8.6771280e-01, 3.5926512e-01, 4.1617877e-03, ..., 9.5753856e-03,
         8.2841674e-03, 9.5275557e-01],
        [7.9850811e-01, 3.6502767e-01, 1.2566844e-03, ..., 9.2500202e-02,
         2.6666848e-02, 8.7240314e-01],
        ...,
        [9.4344538e-01, 7.6549500e-01, 2.2580901e-03, ..., 3.3048126e-01,
         5.8149064e-01, 8.4905243e-01],
        [9.9356246e-01, 8.8543540e-01, 1.4172160e-02, ..., 2.6118333e-04,
         3.0170130e-02, 8.9497721e-01],
        [8.9524102e-01, 4.9251071e-01, 5.6650336e-03, ..., 7.3634677e-02,
         5.1417530e-01, 9.0043950e-01]], dtype=float32)]
```

```
pred=[]
for i in all_predictions[0]:
  pred.append(i.tolist())
```

```
correlation={}
total_score=0
for i in target_cols:
  a=spearmanr(df_test[i].values,df_pred[i].values)
  correlation.update({i:a[0]})
  if  math.isnan(a[0]):
    pass
  else:
    total_score=total_score+a[0]
```

```
final=total_score/30
print(final)
```

```
0.3574074033333333
```

8 Data Pre-Processing for LSTM (GloVe Embedding) Model

For the pre-processing the text data is converted into specific tokens with the help of keras preprocessing library then these tokens are mapped with the pre trained GloVe token file which is downloaded from the above link.

```
[ ] all_df = train_df.append(test_df, sort=False)
```

joing question title, question body and answer together

```
ques_title = list(all_df['question_title'])
ques_body = list(all_df['question_body'])
answer = list(all_df['answer'])
all_text = [tit+body+ans for tit, body,ans in zip(ques_title,ques_body, answer)]
print(all_text[1])
```

```
What is the distinction between a city and a sprawl/metroplex... between downtown and a commercial district?I am trying

Per p 15, a sprawl is a plex, a plex is a "metropolitan complex, short for metroplex". Per Google a metroplex is " a ve
It might be helpful to look into the definition of spam zone:

(p.216) spam zone: An area flooded with invasive and/or viral AR advertising, causing noise.

Because a metroplex has so many marketing targets, it seems a safe assumption that marketers would drown the plex with
```

Converting text to tokens

```
[ ] tok = text.Tokenizer()
    tok.fit_on_texts(all_text)
    X_text = tok.texts_to_sequences(all_text)
    X_text = sequence.pad_sequences(X_text, 300)
```

```
[ ] word_index = tok.word_index
    print('Found %s unique tokens.' % len(word_index))
```

```
Found 59877 unique tokens.
```

Maping the tokens in the dataset with the pre GloVe file token

```
embeddings_index = {}
f = open(os.path.join('/content/drive/My Drive/ThesisProject/glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))
```

```
Found 400000 word vectors.
```

9 LSTM Model Training

Once the text is converted and mapped to the pre-trained GloVe vectors now the model is created which is a combination of LSTM and CNN architecture inspired from the research by (Lee and Dernoncourt, 2016) to get the optimal performance.

```
from keras.initializers import Constant
def bilstm_conv_model():

    inp = Input((X_train.shape[1],))

    embed = Embedding(num_words, embedding_dim,embeddings_initializer=Constant(embedding_matrix),
                      input_length=X_train.shape[1],trainable=True)(inp)

    bilstm = Bidirectional(LSTM(64, return_sequences=True, dropout=0.2, recurrent_dropout=0.2))(embed)

    conv_1 = Conv1D(128, 3, strides=2, padding='valid', use_bias=False)(bilstm)
    conv_1 = BatchNormalization()(conv_1)
    conv_1 = Activation('relu')(conv_1)
    conv_1 = Dropout(0.2)(conv_1)

    conv_2 = Conv1D(256, 3, strides=2, padding='valid', use_bias=False)(conv_1)
    conv_2 = BatchNormalization()(conv_2)
    conv_2 = Activation('relu')(conv_2)
    conv_2 = Dropout(0.3)(conv_2)

    conv_3 = Conv1D(512, 3, strides=2, padding='valid', use_bias=False)(conv_2)
    conv_3 = BatchNormalization()(conv_3)
    conv_3 = Activation('relu')(conv_3)
    conv_3 = Dropout(0.4)(conv_3)

    conv_4 = Conv1D(1024, 3, strides=2, padding='valid', use_bias=False)(conv_3)
    conv_4 = BatchNormalization()(conv_4)
    conv_4 = Activation('relu')(conv_4)
    conv_4 = Dropout(0.5)(conv_4)
    avg_pool = GlobalAveragePooling1D()(conv_4)

    dense = Dense(256, activation='relu')(avg_pool)
    dense = Dropout(0.2)(dense)

    out = Dense(30, activation='sigmoid')(dense)

    model = Model(inputs=inp, outputs=out)
    #print (model.summary())

    return model
```

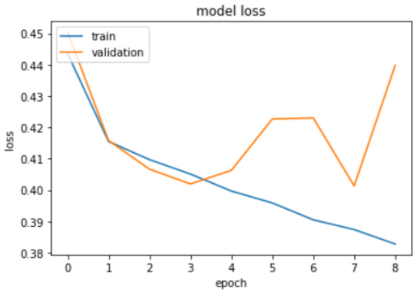
```
model.summary()
```

```
Model: "functional_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 300)]             0         
_________________________________________________________________
embedding (Embedding)        (None, 300, 100)          5618400   
_________________________________________________________________
bidirectional (Bidirectional (None, 300, 128)          84480     
_________________________________________________________________
conv1d (Conv1D)              (None, 149, 128)          49152     
_________________________________________________________________
batch_normalization (BatchNo (None, 149, 128)          512       
_________________________________________________________________
activation (Activation)      (None, 149, 128)          0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 149, 128)          0         
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 74, 256)           98304     
_________________________________________________________________
batch_normalization_1 (Batch (None, 74, 256)           1024      
_________________________________________________________________
activation_1 (Activation)    (None, 74, 256)           0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 74, 256)           0         
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 36, 512)           393216    
_________________________________________________________________
batch_normalization_2 (Batch (None, 36, 512)           2048      
_________________________________________________________________
activation_2 (Activation)    (None, 36, 512)           0         
```

```
_________________________________________________________________
dropout_5 (Dropout)          (None, 36, 512)           0         
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 17, 1024)          1572864   
_________________________________________________________________
batch_normalization_3 (Batch (None, 17, 1024)          4096      
_________________________________________________________________
activation_3 (Activation)    (None, 17, 1024)          0         
_________________________________________________________________
dropout_6 (Dropout)          (None, 17, 1024)          0         
_________________________________________________________________
global_average_pooling1d_3 ( (None, 1024)              0         
_________________________________________________________________
dense (Dense)                (None, 256)               262400    
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 30)                7710      
=================================================================
Total params: 8,094,206
Trainable params: 8,090,366
Non-trainable params: 3,840
_________________________________________________________________
```

```
all_predictions = []

kf = KFold(n_splits=2).split(X_train)

for fold, (tr_idx, val_idx) in enumerate(kf):

    K.clear_session()

    X_tr = X_train[tr_idx]
    y_tr = y_train[tr_idx]
    X_val = X_train[val_idx]
    y_val = y_train[val_idx]

    model = bilstm_conv_model()
    model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0005))
    history=model.fit(X_tr, y_tr,
                      epochs=15,
                      batch_size=32,
                      validation_data=(X_val, y_val),
                      callbacks=[SpearmanRhoCallback(training_data=(X_tr, y_tr),
                                                     validation_data=(X_val, y_val), patience=2)])
    all_predictions.append(model.predict(X_test))

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
Epoch 5/15
val_spearman-rho: 0.2528
95/95 [==============================] - 156s 2s/step - loss: 0.3997 - val_loss: 0.4063
Epoch 6/15
val_spearman-rho: 0.2483
95/95 [==============================] - 154s 2s/step - loss: 0.3959 - val_loss: 0.4227
Epoch 7/15
val_spearman-rho: 0.2581
95/95 [==============================] - 156s 2s/step - loss: 0.3905 - val_loss: 0.4230
Epoch 8/15
val_spearman-rho: 0.2682
95/95 [==============================] - 157s 2s/step - loss: 0.3874 - val_loss: 0.4013
Epoch 9/15
95/95 [==============================] - ETA: 0s - loss: 0.3828Epoch 00008: early stopping Threshold
val_spearman-rho: 0.2512
95/95 [==============================] - 157s 2s/step - loss: 0.3828 - val_loss: 0.4398
```




```
[ ] all_predictions=[]
    all_predictions.append(model.predict(xtest))
```

```
[ ] pred=[]
    for i in all_predictions[0]:
      pred.append(i.tolist())
```

```
[ ] df_pred=pd.DataFrame(pred)
    df_test=pd.DataFrame(ytest)
```

```
[ ] df_test.columns=target_cols
    df_pred.columns=target_cols
```

```
import math
correlation={}
total_score=0
for i in target_cols:
  a=spearmanr(df_test[i].values,df_pred[i].values)
  correlation.update({i:a[0]})
  if  math.isnan(a[0]):
    pass
  else:
    total_score=total_score+a[0]
```

```
final_score=total_score/30
print(final_score)
```

```
0.23829134118080625
```

The validation score while training the model is 0.26 and the test score of the model is 0.23 which is not comparable with the BERT model score which we got earlier.

10 Pre Processing The Data for Universal Sentence Encoder

Same as word embedding model the text in the dataset is converted into vectors the only difference is that the complete sentence is converted into a vector for rather than the word.

```
embeddings_train = {}
embeddings_test = {}
for text in ['question_title', 'question_body', 'answer']:
    print(text)
    train_text = train_df[text].str.replace('?', '.').str.replace('!', '.').tolist()

    curr_train_emb = []
    batch_size = 4
    ind = 0
    while ind*batch_size < len(train_text):
        curr_train_emb.append(embed(train_text[ind*batch_size: (ind + 1)*batch_size])["outputs"].numpy())
        ind += 1

    embeddings_train[text + '_embedding'] = np.vstack(curr_train_emb)

del embed
K.clear_session()
gc.collect()
```

```
l2_dist = lambda x, y: np.power(x - y, 2).sum(axis=1)
cos_dist = lambda x, y: (x*y).sum(axis=1)
l1_dist = lambda x, y: np.abs(x - y).sum(axis=1)

dist_features_train = np.array([
    l1_dist(embeddings_train['question_title_embedding'], embeddings_train['answer_embedding']),
    l1_dist(embeddings_train['question_body_embedding'], embeddings_train['answer_embedding']),
    l1_dist(embeddings_train['question_body_embedding'], embeddings_train['question_title_embedding']),
    l2_dist(embeddings_train['question_title_embedding'], embeddings_train['answer_embedding']),
    l2_dist(embeddings_train['question_body_embedding'], embeddings_train['answer_embedding']),
    l2_dist(embeddings_train['question_body_embedding'], embeddings_train['question_title_embedding']),
    cos_dist(embeddings_train['question_title_embedding'], embeddings_train['answer_embedding']),
```

```
    cos_dist(embeddings_train['question_body_embedding'], embeddings_train['answer_embedding']),
    cos_dist(embeddings_train['question_body_embedding'], embeddings_train['question_title_embedding'])
]).T


X_train_universal = np.hstack([item for k, item in embeddings_train.items()] + [dist_features_train])


del dist_features_train
gc.collect()
```

11 Universal Sentence Encoder Training

Once the sentences in the dataset are converted into vectors these vectors are trained into a neural net model with one input layer 2 hidden layers and an output layer with sigmoid activation function.

```python
def universal_model():

    inp = Input((X_train_universal.shape[1],))

    dense = Dense(512, activation='relu')(inp)
    dense = Dropout(0.2)(dense)
    out = Dense(30, activation='sigmoid')(dense)

    model = Model(inputs=inp, outputs=out)
    #print (model.summary())

    return model
```

```python
model = universal_model()
model.summary()
```

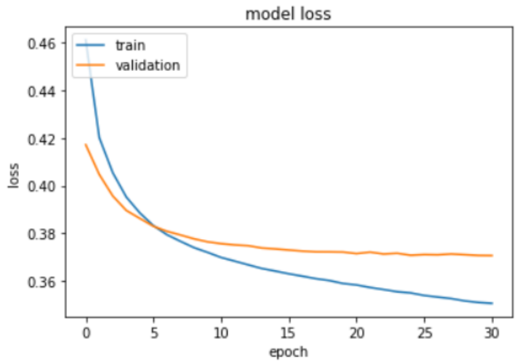
```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 1545)]            0
_________________________________________________________________
dense (Dense)                (None, 512)               791552
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 30)                15390
=================================================================
Total params: 806,942
Trainable params: 806,942
Non-trainable params: 0
```

```python
all_predictions = []

uni_model = universal_model()
uni_model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=0.0001))
history=uni_model.fit(xtrain, ytrain,
                      epochs=200,
                      batch_size=16,
                      validation_data=(xval, yval),
                      callbacks=[SpearmanRhoCallback(training_data=(xtrain, ytrain),
                                                     validation_data=(xval, yval), patience=5)])
```

```
Epoch 1/200
val_spearman-rho: 0.1708
313/313 [==============================] - 1s 4ms/step - loss: 0.4612 - val_loss: 0.4172
Epoch 2/200
val_spearman-rho: 0.2356
313/313 [==============================] - 1s 4ms/step - loss: 0.4201 - val_loss: 0.4047
Epoch 3/200
val_spearman-rho: 0.2715
```

```
Epoch 4/200
val_spearman-rho: 0.2955
313/313 [==============================] - 1s 4ms/step - loss: 0.3952 - val_loss: 0.3896
Epoch 5/200
val_spearman-rho: 0.3136
313/313 [==============================] - 1s 4ms/step - loss: 0.3885 - val_loss: 0.3862
Epoch 6/200
val_spearman-rho: 0.3287
313/313 [==============================] - 1s 4ms/step - loss: 0.3832 - val_loss: 0.3829
Epoch 7/200
val_spearman-rho: 0.3387
313/313 [==============================] - 1s 4ms/step - loss: 0.3794 - val_loss: 0.3808
Epoch 8/200
val_spearman-rho: 0.3478
313/313 [==============================] - 1s 4ms/step - loss: 0.3766 - val_loss: 0.3793
Epoch 9/200
val_spearman-rho: 0.3549
313/313 [==============================] - 1s 4ms/step - loss: 0.3740 - val_loss: 0.3776
Epoch 10/200
val_spearman-rho: 0.3605
313/313 [==============================] - 1s 4ms/step - loss: 0.3720 - val_loss: 0.3764
Epoch 11/200
val_spearman-rho: 0.3641
313/313 [==============================] - 1s 4ms/step - loss: 0.3698 - val_loss: 0.3756
Epoch 12/200
val_spearman-rho: 0.3677
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```




```
all_predictions.append(uni_model.predict(xtest))
```

```
correlation={}
Total_score=0
for i in target_cols:
  a=spearmanr(df_test[i].values,df_pred[i].values)
  correlation.update({i:a[0]})
  if  math.isnan(a[0]):
    pass
  else:
    Total_score=Total_score+a[0]
```

```
correlation
```

```
{'answer_helpful': 0.1685441968502061,
 'answer_level_of_information': 0.4767247179160893,
 'answer_plausible': 0.12449824678857152,
 'answer_relevance': 0.14108906937006097,
 'answer_satisfaction': 0.3634547056271413,
 'answer_type_instructions': 0.7043115034840637,
 'answer_type_procedure': 0.22026897151383026,
 'answer_type_reason_explanation': 0.6388113589582819,
 'answer_well_written': 0.0751866505944764,
 'question_asker_intent_understanding': 0.3780963830461222,
 'question_body_critical': 0.6357758160804492,
 'question_conversational': 0.3279815946476386,
 'question_expect_short_answer': 0.2621870332710231,
 'question_fact_seeking': 0.3000490611552693,

 'question_has_commonly_accepted_answer': 0.4374809843973091,
 'question_interestingness_others': 0.35310592967434995,
 'question_interestingness_self': 0.4820770453074145,
 'question_multi_intent': 0.44263126042936507,
 'question_not_really_a_question': 0.059352818014821625,
 'question_opinion_seeking': 0.3830700422583338,
 'question_type_choice': 0.6029819667959766,
 'question_type_compare': 0.38779196860433973,
 'question_type_consequence': 0.19181927657261472,
 'question_type_definition': 0.38277401057634575,
 'question_type_entity': 0.46041495269390176,
 'question_type_instructions': 0.7252251886819383,
 'question_type_procedure': 0.32026566736854234,
 'question_type_reason_explanation': 0.6343764596541136,
 'question_type_spelling': 0.5433222111,
 'question_well_written': 0.5223988235916301}
```

```
final_score=Total_score/30
print(final_score)
```

```
0.37342485679747406
```

The score for the Universal Sentence Encoder for validation training is 0.381 and the test score is 0.37, which outperforms the BERT and LSTM model. Even while going through the individual correlation score of each attribute, it is clearly visible that Universal Sentence Encoder model is able better understand the subjective aspect of the given question answer pairs.

12 Extra Experiments with the BERT model to Increase the Performance

To increase the performance of the model a hypothesis was assumed that model was not giving optimum performance because of different categories of the data. To overcome the challenge the dataset was divided into two separate groups each having similar categories. Technology, StackOverflow and Science were kept together whereas life arts and culture were kept together in separate database. The BERT model was trained again with only taking the technology group and the readings were taken. While going through the results we could make out the spearman correlation which was 0.37 at the time of training complete dataset is reduced to 0.30 when taking sub set of the dataset, thus proving the hypothesis wrong.

sub domains science, stackoverflow and technology were taken together

```
]: train_df=train_df[train_df.category.isin(['SCIENCE','STACKOVERFLOW','TECHNOLOGY'])]
   test_df=test_df[test_df.category.isin(['SCIENCE','STACKOVERFLOW','TECHNOLOGY'])]
```

```
all_predictions = []
gkf = GroupKFold(n_splits=5).split(X=train_df.question_body, groups=train_df.question_body)

for fold, (tr_idx, val_idx) in enumerate(gkf):

    K.clear_session()

    X_tr = [train_inputs[i][tr_idx] for i in range(3)]
    y_tr = y_train[tr_idx]
    X_val = [train_inputs[i][val_idx] for i in range(3)]
    y_val = y_train[val_idx]

    model = bert_model()
    model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=0.00005))
    model.fit(X_tr, y_tr, epochs=4, batch_size=8, validation_data=(X_val, y_val), callbacks=[SpearmanRhoCallback(training_data=(X
    all_predictions.append(model.predict(test_inputs))

    if fold==2: break

K.clear_session()
model = bert_model()
model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(lr=0.00005))
model.fit(train_inputs, y_train, epochs=4, batch_size=8)
all_predictions.append(model.predict(test_inputs))
```

```
Epoch 1/4
val_spearman-rho: 0.3081
441/441 [==============================] - 286s 649ms/step - loss: 0.3993 - val_loss: 0.3779
Epoch 2/4
val_spearman-rho: 0.3301
441/441 [==============================] - 285s 646ms/step - loss: 0.3745 - val_loss: 0.3731
Epoch 3/4
val_spearman-rho: 0.335
```

13 References

Lee, J. Y. and Dernoncourt, F. (2016) ‘Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks’, *arXiv:1603.03827 [cs, stat]*. Available at: http://arxiv.org/abs/1603.03827 (Accessed: 31 July 2020).