In [ ]:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
import pickle
import re
from scipy.stats import spearmanr
from tqdm import tqdm
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

```
Using TensorFlow backend.
```

In [ ]:

```
train = pd.read_pickle('./train')
train.head()
```

Out[2]:

	qa_id	question_title	question_body	question_user_name	question
0	0	what am i losing when using extension tubes in...	after playing around with macro photography on...	ysap	https://photo.stackexchange.com
1	1	what is the distinction between a city and a s...	i am trying to understand what kinds of places...	russellpierce	https://rpg.stackexchange.com
2	2	maximum protusion length for through hole comp...	i am working on a pcb that has through hole co...	Joe Baker	https://electronics.stackexchange.com/
3	3	can an affidavit be used in beit din	an affidavit from what i understand is basical...	Scimonster	https://judaism.stackexchange.com
4	5	how do you make a binary image in photoshop	i am trying to make a binary image i want more...	leigero	https://graphicdesign.stackexchange.c

In [ ]:

```
test = pd.read_pickle('./test')
test.head()
```

Out[3]:

	qa_id	question_title	question_body	question_user_name	ques
0	39	will leaving corpses lying around upset my pri...	i see questionsinformation online about how to...	Dylan	https://gaming.stackexchange.
1	46	url link to feature image in the portfolio	i am new to wordpress i have issue with featur...	Anu	https://wordpress.stackexchange.
2	70	is accuracy recoil or bullet spread affected b...	to experiment i started a bot game toggled inv...	Konsta	https://gaming.stackexchange.
3	132	suddenly got an io error from my external hdd	i have used my raspberry pi as a torrent serve...	robbannn	https://raspberrypi.stackexchange.
4	200	passenger name flight booking passenger only...	i have bought delhi london return flights for ...	Amit	https://travel.stackexchange.

In [ ]:

```
X = pd.DataFrame()
for i in test.columns:
    X[i] = train[i]
```

In [ ]:

```
X.columns
```

Out[5]:

```
Index(['qa_id', 'question_title', 'question_body', 'question_user_name',
       'question_user_page', 'answer', 'answer_user_name', 'answer_user_pag
e',
       'url', 'category', 'host'],
      dtype='object')
```

In [ ]:

```
features = ['question_title', 'question_body', 'answer', 'question_user_name',
       'question_user_page', 'answer_user_name', 'answer_user_page',
       'url', 'category', 'host']
X[features]
```

Out[6]:

	question_title	question_body	answer	question_user_name	
0	what am i losing when using extension tubes in...	after playing around with macro photography on...	i just got extension tubes so here is the skin...	ysap	https://photo.stacke
1	what is the distinction between a city and a s...	i am trying to understand what kinds of places...	it might be helpful to look into the definitio...	russellpierce	https://rpg.stacke
2	maximum protusion length for through hole comp...	i am working on a pcb that has through hole co...	do you even need grooves we make several pro...	Joe Baker	https://electronics.stackex
3	can an affidavit be used in beit din	an affidavit from what i understand is basical...	sending an affidavit it is a dispute between...	Scimonster	https://judaism.stacke
4	how do you make a binary image in photoshop	i am trying to make a binary image i want more...	check out image trace in adobe illustrator i...	leigero	https://graphicdesign.sta
...	...	...	...	...	
6074	using a ski helmet for winter biking	i am curious if anyone uses a skiing helmet fo...	if you are thinking about wearing a ski helmet...	sixtyfootersdude	https://bicycles.stack
6075	adjustment to road bike brakes for high grade ...	i have a road bike with a front brake that wea...	you can replace the pads as stated elsewhere...	ash	https://bicycles.stackex
6076	suppress file truncated messages when using tail	i am tailing a log file using tail f messages...	maybe help if can be fixes origin of this erro...	Maneating Koala	https://unix.stackex
6077	when should a supervisor be a co author	what are people is views on this to be speci...	as a non mathematician i am somewhat mystified...	MrB	https://mat
6078	why are there so many different types of screw...	newbie question why is it that there is a baz...	first i really like eric is answer for practic...	Doug T.	https://diy.stack

6079 rows × 10 columns

In [ ]:

```
y = pd.DataFrame()
for i in train.columns:
    if i not in X.columns:
        y[i] = train[i]
```

In [ ]:

```
y.columns
```

Out[8]:

```
Index(['question_asker_intent_understanding', 'question_body_critical',
       'question_conversational', 'question_expect_short_answer',
       'question_fact_seeking', 'question_has_commonly_accepted_answer',
       'question_interestingness_others', 'question_interestingness_self',
       'question_multi_intent', 'question_not_really_a_question',
       'question_opinion_seeking', 'question_type_choice',
       'question_type_compare', 'question_type_consequence',
       'question_type_definition', 'question_type_entity',
       'question_type_instructions', 'question_type_procedure',
       'question_type_reason_explanation', 'question_type_spelling',
       'question_well_written', 'answer_helpful',
       'answer_level_of_information', 'answer_plausible', 'answer_relevanc
e',
       'answer_satisfaction', 'answer_type_instructions',
       'answer_type_procedure', 'answer_type_reason_explanation',
       'answer_well_written'],
      dtype='object')
```

In [ ]:

```
words = train['question_title'].apply(lambda x : len(x.split(' ')))
words = sorted(words.values)
for i in range(91,100):

    print("{} percentage question_title text contains words less than : {}".format(i,words[
```

```
91 percentage question_title text contains words less than : 16
92 percentage question_title text contains words less than : 17
93 percentage question_title text contains words less than : 17
94 percentage question_title text contains words less than : 17
95 percentage question_title text contains words less than : 18
96 percentage question_title text contains words less than : 19
97 percentage question_title text contains words less than : 20
98 percentage question_title text contains words less than : 21
99 percentage question_title text contains words less than : 23
```

In [ ]:

```
words = train['question_body'].apply(lambda x : len(x.split(' ')))
words = sorted(words.values)
for i in range(91,100):

    print("{} percentage question_body text contains words less than : {}".format(i,words[r
```

```
91 percentage question_body text contains words less than : 434
92 percentage question_body text contains words less than : 460
93 percentage question_body text contains words less than : 508
94 percentage question_body text contains words less than : 557
95 percentage question_body text contains words less than : 613
96 percentage question_body text contains words less than : 720
97 percentage question_body text contains words less than : 883
98 percentage question_body text contains words less than : 1072
99 percentage question_body text contains words less than : 1489
```

In [ ]:

```
words = train['answer'].apply(lambda x : len(x.split(' ')))
words = sorted(words.values)
for i in range(91,100):

    print("{} percentage answer text contains words less than : {}".format(i,words[round((l
```

```
91 percentage answer text contains words less than : 416
92 percentage answer text contains words less than : 442
93 percentage answer text contains words less than : 473
94 percentage answer text contains words less than : 509
95 percentage answer text contains words less than : 568
96 percentage answer text contains words less than : 638
97 percentage answer text contains words less than : 724
98 percentage answer text contains words less than : 843
99 percentage answer text contains words less than : 1237
```

In [ ]:

```
X_train, X_cv, y_train, y_cv = train_test_split(X[features], y.values, test_size=0.20)
```

In [ ]:

```
import tensorflow as tf
import keras
import os
import random as rn
import tensorflow_hub as hub
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, Flatten,concatenate,Input, Dropout,Batc
from tensorflow.keras import backend as K
from tensorflow.keras.callbacks import EarlyStopping,ModelCheckpoint,Callback
np.random.seed(42)

tf.random.set_seed(20)

rn.seed(12)
```

In [ ]:

```
tf.keras.backend.clear_session()

max_seq_length = 25

input_word_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="input

input_mask = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="input_mas

segment_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="segment_

bert_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1
pooled_output, sequence_output = bert_layer([input_word_ids, input_mask, segment_ids])

bert_model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=pooled_output)
```

In [ ]:

```
vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()
```

In [ ]:

```
!pip install sentencepiece
```

```
Collecting sentencepiece
  Downloading https://files.pythonhosted.org/packages/d4/a4/d0a884c4300004a7
8cca907a6ff9a5e9fe4f090f5d95ab341c53d28cbc58/sentencepiece-0.1.91-cp36-cp36m
-manylinux1_x86_64.whl (https://files.pythonhosted.org/packages/d4/a4/d0a884
c4300004a78cca907a6ff9a5e9fe4f090f5d95ab341c53d28cbc58/sentencepiece-0.1.91-
cp36-cp36m-manylinux1_x86_64.whl) (1.1MB)
     |████████████████████████████████| 1.1MB 2.7MB/s
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.91
```

In [ ]:

```
import tokenization
```

In [ ]:

```
tokenizer = tokenization.FullTokenizer(vocab_file,do_lower_case)
```

In [ ]:

```
X_train_qt_tokens = np.zeros(shape=(X_train.shape[0],25))
X_train_qt_mask = np.zeros(shape=(X_train.shape[0],25))
X_train_qt_segment = np.zeros(shape=(X_train.shape[0],25))
X_cv_qt_tokens = np.zeros(shape=(X_cv.shape[0],25))
X_cv_qt_mask = np.zeros(shape=(X_cv.shape[0],25))
X_cv_qt_segment = np.zeros(shape=(X_cv.shape[0],25))
```

In [ ]:

```
for i in range(X_train.shape[0]):
    tokens = tokenizer.tokenize(X_train.values[i][0])
    if len(tokens) >= max_seq_length-2:
        tokens = tokens[0:(max_seq_length-2)]
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_train_qt_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_train_qt_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        X_train_qt_segment[i] = np.array([0]*max_seq_length)
    else:
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_train_qt_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        tokens = tokens + ['[PAD]']*(max_seq_length-len(tokens))
        X_train_qt_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_train_qt_segment[i] = np.array([0]*max_seq_length)
```

In [ ]:

```
for i in range(X_cv.shape[0]):
    tokens = tokenizer.tokenize(X_cv.values[i][0])
    if len(tokens) >= max_seq_length-2:
        tokens = tokens[0:(max_seq_length-2)]
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_cv_qt_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_cv_qt_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        X_cv_qt_segment[i] = np.array([0]*max_seq_length)
    else:
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_cv_qt_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        tokens = tokens + ['[PAD]']*(max_seq_length-len(tokens))
        X_cv_qt_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_cv_qt_segment[i] = np.array([0]*max_seq_length)
```

In [ ]:

```
X_train_qt_pooled_output = bert_model.predict([X_train_qt_tokens.astype('int32'), X_train_q
X_cv_qt_pooled_output = bert_model.predict([X_cv_qt_tokens.astype('int32'), X_cv_qt_mask.as
```

In [ ]:

```
tf.keras.backend.clear_session()

max_seq_length = 512

input_word_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="input

input_mask = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="input_mas

segment_ids = tf.keras.layers.Input(shape=(max_seq_length,), dtype=tf.int32, name="segment_

bert_layer = hub.KerasLayer("https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1
pooled_output, sequence_output = bert_layer([input_word_ids, input_mask, segment_ids])

bert_model = Model(inputs=[input_word_ids, input_mask, segment_ids], outputs=pooled_output)
```

In [ ]:

```
#getting Vocab file
vocab_file = bert_layer.resolved_object.vocab_file.asset_path.numpy()
do_lower_case = bert_layer.resolved_object.do_lower_case.numpy()

tokenizer = tokenization.FullTokenizer(vocab_file,do_lower_case)
```

In [ ]:

```
X_train_q_tokens = np.zeros(shape=(X_train.shape[0],512))
X_train_q_mask = np.zeros(shape=(X_train.shape[0],512))
X_train_q_segment = np.zeros(shape=(X_train.shape[0],512))
X_cv_q_tokens = np.zeros(shape=(X_cv.shape[0],512))
X_cv_q_mask = np.zeros(shape=(X_cv.shape[0],512))
X_cv_q_segment = np.zeros(shape=(X_cv.shape[0],512))
```

In [ ]:

```
for i in range(X_train.shape[0]):
    tokens = tokenizer.tokenize(X_train.values[i][1])
    if len(tokens) >= max_seq_length-2:
        tokens = tokens[0:(max_seq_length-2)]
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_train_q_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_train_q_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        X_train_q_segment[i] = np.array([0]*max_seq_length)
    else:
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_train_q_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        tokens = tokens + ['[PAD]']*(max_seq_length-len(tokens))
        X_train_q_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_train_q_segment[i] = np.array([0]*max_seq_length)
```

In [ ]:

```
for i in range(X_cv.shape[0]):
    tokens = tokenizer.tokenize(X_cv.values[i][1])
    if len(tokens) >= max_seq_length-2:
        tokens = tokens[0:(max_seq_length-2)]
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_cv_q_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_cv_q_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        X_cv_q_segment[i] = np.array([0]*max_seq_length)
    else:
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_cv_q_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        tokens = tokens + ['[PAD]']*(max_seq_length-len(tokens))
        X_cv_q_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_cv_q_segment[i] = np.array([0]*max_seq_length)
```

In [ ]:

```
X_train_q_pooled_output = bert_model.predict([X_train_q_tokens.astype('int32'), X_train_q_m
X_cv_q_pooled_output = bert_model.predict([X_cv_q_tokens.astype('int32'), X_cv_q_mask.astyp
```

In [ ]:

```
X_train_a_tokens = np.zeros(shape=(X_train.shape[0],512))
X_train_a_mask = np.zeros(shape=(X_train.shape[0],512))
X_train_a_segment = np.zeros(shape=(X_train.shape[0],512))
X_cv_a_tokens = np.zeros(shape=(X_cv.shape[0],512))
X_cv_a_mask = np.zeros(shape=(X_cv.shape[0],512))
X_cv_a_segment = np.zeros(shape=(X_cv.shape[0],512))
```

In [ ]:

```
for i in range(X_train.shape[0]):
    tokens = tokenizer.tokenize(X_train.values[i][2])
    if len(tokens) >= max_seq_length-2:
        tokens = tokens[0:(max_seq_length-2)]
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_train_a_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_train_a_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        X_train_a_segment[i] = np.array([0]*max_seq_length)
    else:
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_train_a_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        tokens = tokens + ['[PAD]']*(max_seq_length-len(tokens))
        X_train_a_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_train_a_segment[i] = np.array([0]*max_seq_length)
```

In [ ]:

```
for i in range(X_cv.shape[0]):
    tokens = tokenizer.tokenize(X_cv.values[i][2])
    if len(tokens) >= max_seq_length-2:
        tokens = tokens[0:(max_seq_length-2)]
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_cv_a_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_cv_a_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        X_cv_a_segment[i] = np.array([0]*max_seq_length)
    else:
        tokens = ['[CLS]',*tokens,'[SEP]']
        X_cv_a_mask[i] = np.array([1]*len(tokens)+[0]*(max_seq_length-len(tokens)))
        tokens = tokens + ['[PAD]']*(max_seq_length-len(tokens))
        X_cv_a_tokens[i] = np.array(tokenizer.convert_tokens_to_ids(tokens))
        X_cv_a_segment[i] = np.array([0]*max_seq_length)
```

In [ ]:

```
X_train_a_pooled_output = bert_model.predict([X_train_a_tokens.astype('int32'), X_train_a_m
X_cv_a_pooled_output = bert_model.predict([X_cv_a_tokens.astype('int32'), X_cv_a_mask.astyp
```

In [ ]:

```
class SpearmanCallback(Callback):
    def __init__(self, validation_data, patience, model_name):
        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_epoch_end(self, epoch, logs={}):
        y_pred_val = self.model.predict(self.x_val)
        rho_val = np.mean([spearmanr(self.y_val[:, ind], y_pred_val[:, ind]+ np.random.norm
        if rho_val >= self.value:
            self.value = rho_val
            self.bad_epochs = 0
            self.model.save_weights(self.model_name)
        else:
            self.bad_epochs += 1
        if self.bad_epochs >= self.patience:
            print("early stopping Threshold")
            self.model.stop_training = True
        print('\nbad: {}'.format(self.bad_epochs))
        print('\nval_spearman-corr: %s' % (str(round(rho_val, 4))), end=100*' '+'\n')
        return rho_val
```

Bert Model

In [ ]:

```
K.clear_session()
input_1 = Input(shape=(X_train_qt_pooled_output.shape[1]))
qt = Dense(66,activation = 'relu',kernel_initializer=tf.keras.initializers.glorot_uniform(s

input_2 = Input(shape=(X_train_q_pooled_output.shape[1]))
q = Dense(66,activation = 'relu',kernel_initializer=tf.keras.initializers.glorot_uniform(se

input_3 = Input(shape=(X_train_a_pooled_output.shape[1]))
a = Dense(66,activation = 'relu',kernel_initializer=tf.keras.initializers.glorot_uniform(se

concat = concatenate([qt, q, a])

output = Dense(30, activation='sigmoid',kernel_initializer=tf.keras.initializers.glorot_uni

model_bert = Model(inputs=[input_1, input_2, input_3], outputs=output)

model_bert.summary()
```

```
Model: "model"
__________________________________________________________________________
_____________________
Layer (type)                 Output Shape              Param #     Connected
to
==========================================================================
=======================
input_1 (InputLayer)         [(None, 768)]             0
__________________________________________________________________________
_____________________
input_2 (InputLayer)         [(None, 768)]             0
__________________________________________________________________________
_____________________
input_3 (InputLayer)         [(None, 768)]             0
__________________________________________________________________________
_____________________
dense (Dense)                (None, 66)                50754       input_1
[0][0]
__________________________________________________________________________
_____________________
dense_1 (Dense)              (None, 66)                50754       input_2
[0][0]
__________________________________________________________________________
_____________________
dense_2 (Dense)              (None, 66)                50754       input_3
[0][0]
__________________________________________________________________________
_____________________
concatenate (Concatenate)    (None, 198)               0           dense[0]
[0]
                                                                   dense_1
[0][0]
                                                                   dense_2
[0][0]
__________________________________________________________________________
_____________________
dense_3 (Dense)              (None, 30)                5970        concatena
te[0][0]
==========================================================================
=======================
Total params: 158,232
```

```
Trainable params: 158,232
Non-trainable params: 0
_________________________________________________________________
_________________________
```

In [ ]:

```
train_data = [X_train_qt_pooled_output,X_train_q_pooled_output,X_train_a_pooled_output]
cv_data = [X_cv_qt_pooled_output,X_cv_q_pooled_output,X_cv_a_pooled_output]
```

In [ ]:

```
model_bert.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),loss='binary_crossentropy')
```

In [ ]:

```
log_dir = os.path.join('model_bert')

tensorboard_callback1 = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1,
                                                       write_graph=True,write_grads=True)
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `
TensorBoard` Callback.

WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `
TensorBoard` Callback.
```

In [ ]:

```
model_bert.fit(train_data, y_train, batch_size=32, epochs=100, verbose=1,
          validation_data=(cv_data, y_cv),
          callbacks=[SpearmanCallback(validation_data=(cv_data, y_cv),patience=5,
                     model_name='best_weight_bert.h5')])
```

```
Epoch 1/100
139/152 [==========================>...] - ETA: 0s - loss: 0.4274
bad: 0

val_spearman-corr: 0.2048
152/152 [==============================] - 1s 7ms/step - loss: 0.4264 - va
l_loss: 0.4130
Epoch 2/100
144/152 [===========================>..] - ETA: 0s - loss: 0.4085
bad: 0

val_spearman-corr: 0.2454
152/152 [==============================] - 1s 5ms/step - loss: 0.4085 - va
l_loss: 0.4080
Epoch 3/100
141/152 [==========================>...] - ETA: 0s - loss: 0.4026
bad: 0

val_spearman-corr: 0.2676
```

In [ ]:

```
%%time
model_bert.load_weights("best_weight_bert.h5")
y_pred_val = model_bert.predict(cv_data)
sp = np.mean([spearmanr(y_cv[:, ind], y_pred_val[:, ind]).correlation for ind in range(y_pr
```

```
CPU times: user 158 ms, sys: 5.29 ms, total: 163 ms
Wall time: 149 ms
```

In [ ]:

```
print("spearman {}".format(sp))
```

```
spearman 0.3445124521318975
```

#Analysis on features

In [304]:

```
y_pred_train = model_bert.predict(train_data)
sp = np.mean([spearmanr(y_train[:, ind], y_pred_train[:, ind]).correlation for ind in range
featurs_names = y.columns
feature_spear_train = []
mae_train = []
for ind in range(y_pred_train.shape[1]):
    val = spearmanr(y_train[:, ind], y_pred_train[:, ind]).correlation
    mae_train.append(bce(y_train[:, ind], y_pred_train[:, ind]).numpy())
    feature_spear_train.append(val)
    print(str(featurs_names[ind]) + "---------------->" + str(val)+"------------->"+str(bce(
```

```
question_asker_intent_understanding---------------->0.4035010077152568-------
------>0.09424638
question_body_critical---------------->0.6269159177965451------------->0.1383
5846
question_conversational---------------->0.4154958547370616------------->0.074
99526
question_expect_short_answer---------------->0.3007511919040694------------->
0.28095523
question_fact_seeking---------------->0.33786445086156497------------->0.2294
8657
question_has_commonly_accepted_answer---------------->0.42589108682037286----
--------->0.22054912
question_interestingness_others---------------->0.379098239585988------------
->0.09993303
question_interestingness_self---------------->0.5050015034850729-------------
>0.120040365
question_multi_intent---------------->0.4013995165144236------------->0.25737
602
question_not_really_a_question---------------->0.09765900613572526-----------
-->0.008653694
question_opinion_seeking---------------->0.4350253467913642------------->0.27
58377
question_type_choice---------------->0.5223510458018923------------->0.250467
75
question_type_compare---------------->0.29544336483055766------------->0.0791
2785
question_type_consequence---------------->0.16010115039501274------------->0.
020160414
question_type_definition---------------->0.3403277849653273------------->0.05
6397885
question_type_entity---------------->0.39458938124972914------------->0.09793
226
question_type_instructions---------------->0.7251678793825055------------->0.
2355821
question_type_procedure---------------->0.3140283769786121------------->0.191
0478
question_type_reason_explanation---------------->0.5631659936839652----------
--->0.26470792
question_type_spelling---------------->0.06818094745654887------------->0.002
4752088
question_well_written---------------->0.502512005632925------------->0.122445
084
answer_helpful---------------->0.24943244875673107------------->0.08904224
answer_level_of_information---------------->0.4231942203394315------------->
0.071755044
answer_plausible---------------->0.1754033711740405------------->0.06670687
answer_relevance---------------->0.18355128925728348------------->0.053134196
```

```
answer_satisfaction--------------->0.3149270300905689------------->0.0994216
8
answer_type_instructions--------------->0.7357066667671042------------->0.22
688371
answer_type_procedure--------------->0.2851081572501178------------->0.16813
335
answer_type_reason_explanation--------------->0.6076529015062466-----------
->0.26695037
answer_well_written--------------->0.20317068120756412------------->0.077300
735
```

In [324]:

```
sort_indexes
```

Out[324]:

```
array([19,  9, 13, 23, 24, 29, 21, 27, 12,  3, 17, 25,  4, 14,  6, 15,  8,
        0,  2, 22,  5, 10, 20,  7, 11, 18, 28,  1, 16, 26])
```

In [306]:

```
#Sorting features on their spearman rank correlation values

sort_indexes = np.argsort(np.array(feature_spear_train))
least_spearman_features_5 = []
for ind in sort_indexes:
    print(str(featurs_names[ind]) + "--------------->" + str(feature_spear_train[ind])+
          "------------->"+str(mae_train[ind]))
```

```
question_type_spelling--------------->0.06818094745654887------------->0.002
4752088
question_not_really_a_question--------------->0.09765900613572526----------
-->0.008653694
question_type_consequence--------------->0.16010115039501274------------->0.
020160414
answer_plausible--------------->0.1754033711740405------------->0.06670687
answer_relevance--------------->0.18355128925728348------------->0.053134196
answer_well_written--------------->0.20317068120756412------------->0.077300
735
answer_helpful--------------->0.24943244875673107------------->0.08904224
answer_type_procedure--------------->0.2851081572501178------------->0.16813
335
question_type_compare--------------->0.29544336483055766------------->0.0791
2785
question_expect_short_answer--------------->0.3007511919040694------------->
0.28095523
question_type_procedure--------------->0.3140283769786121------------->0.191
0478
answer_satisfaction--------------->0.3149270300905689------------->0.0994216
8
question_fact_seeking--------------->0.33786445086156497------------->0.2294
8657
question_type_definition--------------->0.3403277849653273------------->0.05
6397885
question_interestingness_others--------------->0.379098239585988----------
->0.09993303
question_type_entity--------------->0.39458938124972914------------->0.09793
226
question_multi_intent--------------->0.4013995165144236------------->0.25737
602
question_asker_intent_understanding--------------->0.4035010077152568-------
------>0.09424638
question_conversational--------------->0.4154958547370616------------->0.074
99526
answer_level_of_information--------------->0.4231942203394315------------->
0.071755044
question_has_commonly_accepted_answer--------------->0.42589108682037286----
--------->0.22054912
question_opinion_seeking--------------->0.4350253467913642------------->0.27
58377
question_well_written--------------->0.502512005632925------------->0.122445
084
question_interestingness_self--------------->0.5050015034850729------------
>0.120040365
question_type_choice--------------->0.5223510458018923------------->0.250467
75
question_type_reason_explanation--------------->0.5631659936839652----------
--->0.26470792
answer_type_reason_explanation--------------->0.6076529015062466------------
->0.26695037
```

```
question_body_critical--------------->0.6269159177965451------------->0.1383
5846
question_type_instructions--------------->0.7251678793825055------------->0.
2355821
answer_type_instructions--------------->0.7357066667671042------------->0.22
688371
```

This above results shows that for high spearman value the mean absolute value is high.

In [328]:

```
least_spearman_feature_num = sort_indexes[0]
high_spearman_feature_num = sort_indexes[-1]
```

In [330]:

```
list_1 = list(abs(y_train[:, least_spearman_feature_num]-y_pred_train[:, least_spearman_fea
least_spearman_feature_absolute_diff = sum(list_1)

list_2 = list(abs(y_train[:, high_spearman_feature_num]-y_pred_train[:, high_spearman_featu
high_spearman_feature_absolute_diff = sum(list_2)
```

In [331]:

```
print("least spearman feature absolute differance is --->",least_spearman_feature_absolute_
print("high spearman feature absolute differance is --->",high_spearman_feature_absolute_di
```

```
least spearman feature absolute differance is ---> 12.03694015343276
high spearman feature absolute differance is ---> 1103.335360874615
```

The spearman rank correlation coefficient is behaving diferently. For least absolute differance the value is high and for spearman value the absolute differance is high.

Analysis on Input Data

In [246]:

```
list_index = X_train.index     # to get the indexes of the train data
```

In [256]:

```
y_pred_train = model_bert.predict(train_data)
bce = tf.keras.losses.MeanAbsoluteError()
row_train_loss = []
for i in range(y_pred_train.shape[0]):
    val = bce(y_train[i], y_pred_train[i]).numpy()
    row_train_loss.append(val)
    print(str(list_index[i]) + "--------------->"+ str(val))
```

```
4777--------------->0.14261757
4108--------------->0.17923374
2231--------------->0.18018332
5414--------------->0.12943792
535--------------->0.12675454
4897--------------->0.14864625
1196--------------->0.19078495
757--------------->0.18386035
5181--------------->0.12017755
3039--------------->0.1870161
1926--------------->0.122051634
5561--------------->0.13885337
3654--------------->0.09334591
853--------------->0.10752473
4106--------------->0.11482027
4421--------------->0.12707959
4210--------------->0.12376194
4079--------------->0.1311424
5217--------------->0.1268488
3364--------------->0.13972716
```

In [259]:

```
sort_indexes_row = np.argsort(np.array(row_train_loss))
print("low mean absolute error index  ",list_index[sort_indexes_row[0]])
print("high mean absolute error index  ",list_index[sort_indexes_row[-1]])
```

```
low mean absolute error index   85
high mean absolute error index   5690
```

In [261]:

```
print("low mean absolute error  ",row_train_loss[sort_indexes_row[0]])
print("high mean absolute error  ",row_train_loss[sort_indexes_row[-1]])
```

```
low mean absolute error   0.02094543
high mean absolute error   0.3075962
```

In [262]:

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
t1 = Tokenizer(filters = " ")
t1.fit_on_texts(X_train['question_title'])
t2 = Tokenizer(filters = " ")
t2.fit_on_texts(X_train['question_body'])
t3 = Tokenizer(filters = " ")
t3.fit_on_texts(X_train['answer'])
```

In [263]:

```
#distinct words in respective question_title, question_body, answer features
print(len(t1.word_index))
print(len(t2.word_index))
print(len(t3.word_index))
```

```
6842
31125
39035
```

Frequent words

In [281]:

```
def spec_frequent(t):
    a1 = X_train['question_title'][t].split()
    b1 = X_train['question_body'][t].split()
    c1 = X_train['answer'][t].split()
    count_ = 0
    for i in a1:
        if t1.word_counts[i]>500:
            count_ +=1
    print(str(len(a1))+"---------->"+str(count_))

    count_ = 0
    for i in b1:
        if t2.word_counts[i]>500:
            count_ +=1
    print(str(len(b1))+"---------->"+str(count_))

    count_ = 0
    for i in c1:
        if t3.word_counts[i]>500:
            count_ +=1
    print(str(len(c1))+"---------->"+str(count_))
```

In [294]:

```
print("Frequnecy of Most frequent words")
spec_frequent(list_index[sort_indexes_row[0]])
#number of words in data point ------------------> number of frequent words
```

```
Frequnecy of Most frequent words
10---------->3
128---------->71
104---------->60
```

In [293]:

```
print("Frequnecy of Most frequent words")
spec_frequent(list_index[sort_indexes_row[-1]])
#number of words in data point ------------------> number of frequent words
```

```
Frequnecy of Most frequent words
7---------->2
154---------->67
87---------->39
```

Most frequent words are avaliable mostly in equal proportion for high loss and least loss produced data point.

Rare words

In [278]:

```
def spec(t):
    a1 = X_train['question_title'][t].split()
    b1 = X_train['question_body'][t].split()
    c1 = X_train['answer'][t].split()
    count_ = 0
    for i in a1:
        if t1.word_counts[i]<5:
            count_ +=1
    print(str(len(a1))+"---------->"+str(count_))

    count_ = 0
    for i in b1:
        if t2.word_counts[i]<5:
            count_ +=1
    print(str(len(b1))+"---------->"+str(count_))

    count_ = 0
    for i in c1:
        if t3.word_counts[i]<5:
            count_ +=1
    print(str(len(c1))+"---------->"+str(count_))
```

In [291]:

```
print("Frequnecy of rare words")
spec(list_index[sort_indexes_row[0]])
#number of words in data point ------------------> number of rare words
```

```
Frequnecy of rare words
10---------->0
128---------->0
104---------->1
```

In [292]:

```
print("Frequnecy of rare words")
spec(list_index[sort_indexes_row[-1]])
#number of words in data point ------------------> number of rare words
```

```
Frequnecy of rare words
7---------->1
154---------->13
87---------->10
```

Rare words are more in high loss produced data point. Low loss produced data point has only one rare word that to for answer feature.

Concusion:

1. The value absolute diffearnce is we got is high for less spearman value and less absolute differance for high spearman value.This is due to the dataset having not purely regression values. The features are comprised of 3 or 5 or 9 unique values. The values we are predicting are regression values. The spearman has some issue with this type of data set.

2. The input data points are getting high aboslute differance with the predicted values when the data point have high number of rare words in data. If rare words are less then least absolute differance is low means predicted values are nearer to the original values.

In [ ]: