In [ ]:
```
#Import Statements
import pandas as pd
import numpy as np
import gc
import os
from math import floor, ceil
from tqdm.notebook import tqdm

from sklearn.model_selection import GroupKFold
from scipy.stats import spearmanr

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow.keras.backend as K

from transformers import BertTokenizer
from transformers import BertConfig
from transformers import TFBertModel
```

In [ ]:
```
#*================================================Read data and toke
nizer================================================*

#Read tokenizer and data, as well as defining the maximum sequence leng
th that will be used for the input to Bert (maximum is usually 512 toke
ns)

PATH = '../input/google-quest-challenge/'

BERT_PATH = '../input/bert-base-uncased-huggingface-transformer/'
tokenizer = BertTokenizer.from_pretrained(BERT_PATH+'bert-base-uncased-
vocab.txt')

MAX_SEQUENCE_LENGTH = 384

df_train = pd.read_csv(PATH+'train.csv')
df_test = pd.read_csv(PATH+'test.csv')
```

```
df_sub = pd.read_csv(PATH+'sample_submission.csv')
print('train shape =', df_train.shape)
print('test shape =', df_test.shape)

output_categories = list(df_train.columns[11:])
input_categories = list(df_train.columns[[1,2,5]])
print('\noutput categories:\n\t', output_categories)
print('\ninput categories:\n\t', input_categories)
```

Observations

1) Our train dataset consists of 6079 rows and test dataset consists of 476 rows.

2) We have total 30 output labels and 3 input columns i.e. Question Title, Question Body and Answer.

In [ ]:
```
#*=========================================================Preprocessing F
unctions=========================================================*

# These are some functions that will be used to preprocess the raw text
 data into useable Bert inputs.

def _convert_to_transformer_inputs(title, question, answer, tokenizer,
max_sequence_length):
    """Converts tokenized input to ids, masks and segments for transfor
mer (including bert)"""

    def return_id(str1, str2, truncation_strategy, length):

        inputs = tokenizer.encode_plus(str1, str2,
            add_special_tokens=True,
            max_length=length,
            truncation_strategy=truncation_strategy)

        input_ids =  inputs["input_ids"]
        input_masks = [1] * len(input_ids)
        input_segments = inputs["token_type_ids"]
        padding_length = length - len(input_ids)
```

```
        padding_id = tokenizer.pad_token_id
        input_ids = input_ids + ([padding_id] * padding_length)
        input_masks = input_masks + ([0] * padding_length)
        input_segments = input_segments + ([0] * padding_length)

        return [input_ids, input_masks, input_segments]

    input_ids_q, input_masks_q, input_segments_q = return_id(
        title + ' ' + question, None, 'longest_first', max_sequence_len
gth)

    input_ids_a, input_masks_a, input_segments_a = return_id(
        answer, None, 'longest_first', max_sequence_length)

    return [input_ids_q, input_masks_q, input_segments_q,
            input_ids_a, input_masks_a, input_segments_a]

def compute_input_arrays(df, columns, tokenizer, max_sequence_length):
    input_ids_q, input_masks_q, input_segments_q = [], [], []
    input_ids_a, input_masks_a, input_segments_a = [], [], []
    for _, instance in tqdm(df[columns].iterrows()):
        t, q, a = instance.question_title, instance.question_body, inst
ance.answer

        ids_q, masks_q, segments_q, ids_a, masks_a, segments_a = \
        _convert_to_transformer_inputs(t, q, a, tokenizer, max_sequence
_length)

        input_ids_q.append(ids_q)
        input_masks_q.append(masks_q)
        input_segments_q.append(segments_q)

        input_ids_a.append(ids_a)
        input_masks_a.append(masks_a)
        input_segments_a.append(segments_a)

    return [np.asarray(input_ids_q, dtype=np.int32),
            np.asarray(input_masks_q, dtype=np.int32),
            np.asarray(input_segments_q, dtype=np.int32),
```

```
            np.asarray(input_ids_a, dtype=np.int32),
            np.asarray(input_masks_a, dtype=np.int32),
            np.asarray(input_segments_a, dtype=np.int32)]

def compute_output_arrays(df, columns):
    return np.asarray(df[columns])
```

Observations:

1) _convert_to_transformer_inputs is a generic method to convert our dataset into equivalent input that any model of transformer library is expecting.

2) Question body and Question Title are merged into one and Answer is taken as a separate segment.

In [ ]:
```
#*=========================================Model Creation and Metric
 Calculation=======================================*

def compute_spearmanr_ignore_nan(trues, preds):
    rhos = []
    for tcol, pcol in zip(np.transpose(trues), np.transpose(preds)):
        rhos.append(spearmanr(tcol, pcol).correlation)
    return np.nanmean(rhos)


q_id = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)
a_id = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)

q_mask = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)
a_mask = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)

q_atn = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)
a_atn = tf.keras.layers.Input((MAX_SEQUENCE_LENGTH,), dtype=tf.int32)


def create_model():

    config = BertConfig() # print(config) to see settings
```

```
    config.output_hidden_states = False # Set to True to obtain hidden
 states
    # caution: when using e.g. XLNet, XLNetConfig() will automatically
 use xlnet-large config

    # normally ".from_pretrained('bert-base-uncased')", but because of
 no internet, the
    # pretrained model has been downloaded manually and uploaded to kag
gle.
    bert_model = TFBertModel.from_pretrained(
        BERT_PATH+'bert-base-uncased-tf_model.h5', config=config)

    # if config.output_hidden_states = True, obtain hidden states via b
ert_model(...)[-1]
    q_embedding = bert_model(q_id, attention_mask=q_mask, token_type_id
s=q_atn)[0]
    a_embedding = bert_model(a_id, attention_mask=a_mask, token_type_id
s=a_atn)[0]

    q = tf.keras.layers.GlobalAveragePooling1D()(q_embedding)
    a = tf.keras.layers.GlobalAveragePooling1D()(a_embedding)

    x = tf.keras.layers.Concatenate()([q, a])

    x = tf.keras.layers.Dropout(0.2)(x)

    x = tf.keras.layers.Dense(30, activation='sigmoid')(x)

    model = tf.keras.models.Model(inputs=[q_id, q_mask, q_atn, a_id, a_
mask, a_atn,], outputs=x)

    return model
```

Observations:

1) compute_spearmanr_ignore_nan is used to compute the competition metric for the validation set

2) create_model contains the actual architecture that will be used to finetune BERT to our dataset.

In [ ]:
```
#*=====================================Generating Input and Output for
 Traina & Test=====================================*

outputs = compute_output_arrays(df_train, output_categories)
inputs = compute_input_arrays(df_train, input_categories, tokenizer, MA
X_SEQUENCE_LENGTH)
test_inputs = compute_input_arrays(df_test, input_categories, tokenizer
, MAX_SEQUENCE_LENGTH)
```

In [ ]:
```
#*================================================Training, Validation a
nd Testing================================================*

gkf = GroupKFold(n_splits=5).split(X=df_train.question_body, groups=df_
train.question_body)

valid_preds = []
test_preds = []
for fold, (train_idx, valid_idx) in enumerate(gkf):

    # will actually only do 2 folds (out of 5) to manage < 2h
    if fold in [0, 2]:

        train_inputs = [inputs[i][train_idx] for i in range(len(inputs
))]
        train_outputs = outputs[train_idx]

        valid_inputs = [inputs[i][valid_idx] for i in range(len(inputs
))]
        valid_outputs = outputs[valid_idx]

        K.clear_session()
        model = create_model()
        optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5)
        model.compile(loss='binary_crossentropy', optimizer=optimizer)
        model.fit(train_inputs, train_outputs, epochs=3, batch_size=6)
```

```
        # model.save_weights(f'bert-{fold}.h5')
        valid_preds.append(model.predict(valid_inputs))
        test_preds.append(model.predict(test_inputs))

        rho_val = compute_spearmanr_ignore_nan(valid_outputs, valid_pre
ds[-1])
        print('validation score = ', rho_val)
```

Observations:

1) Loops over the folds in gkf and trains each fold for 3 epochs with a learning rate of 3e-5 and batch_size of 6.

2) A simple binary crossentropy is used as the objective loss-function.

In [ ]:
```
#*=============================================Process and Submit Tes
t Predictions=============================================*

df_sub.iloc[:, 1:] = np.average(test_preds, axis=0)

df_sub.to_csv('submission.csv', index=False)
```

Observations:

Average fold predictions, then save as submission.csv