Extraction of the Triggering Causes of a Query Event

MSc Research Project
Data Analytics

Srijon Datta
Student ID: 21225265

School of Computing
National College of Ireland

Supervisor: Prof. Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Srijon Datta
Student ID:	21225265
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Prof. Vladimir Milosavljevic
Submission Due Date:	14/08/2023
Project Title:	Extraction of the Triggering Causes of a Query Event
Word Count:	984
Page Count:	11

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:	Srijon Datta
Date:	18th September 2023

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):	

Extraction of the Triggering Causes of a Query Event

Srijon Datta
21225265

1 Overview

The "Extraction of the Triggering Causes of a Query Event" research project documentation is available here. This document outlines the configuration set up and how to run each module in detail.

2 Hardware & Software Specifications

2.1 Hardware Specification

All the modules of this thesis are run on a remote server whose specification is given as below.

1. Worker Nodes:
   - 12 x Dell R640 - 2 x Intel Xeon Gold 6252 2.1Ghz (24 Core) + 384GB RAM
   - 20 x Dell R640 - 2 x Intel Xeon Gold 6152 2.1GHz (22 Core) + 384GB RAM
   - 12 x Dell C6220 v2 - 2 x Intel E5-2660 v2 2.2 Ghz (10 Core) + 128GB RAM
2. New High-Memory node
   - 1 x Dell R640 (36 cores: 1536GB RAM)
3. GPU Nodes
   - 5 x Dell R740XD each with 2 Nvidia V100 (32GB) : 256GB RAM
   - 2 X Dell R7525 with 2 Nvidia A100 (40GB) : 384GB RAM

2.2 Software Specification

The following software has been used to build the codes and to execute them properly for this piece of research.

1. Integrated Development Environment (IDE):
   - Apache NetBeans 12.6 (For JAVA modules)
   - Pycharm Community Edition (For Python modules)
2. Scripting Languages:

- JDK 1.8 and above
- Python 3.9

3. Cloud Storage:

- Google Drive

4. Other Tools:

- MS-Excel and Google Sheets (To draw the charts)
- MS-PowerPoint (To draw the model diagrams)
- Overleaf (To write the reports and configuration manual)

3 Environment Setup

A dedicated Virtual Environment was created for this work. just verify that the following language packages are compatible with your virtual environment. Please check for conflicts if you wish to use a higher or lower version of any of the following.

3.1 List of Packages

- `JDK 1.8.0 or above`
- `lucene 5.3.1`

Figure 1: JAVA Packages (for InteractionMatrix)

- `conda 4.8.2`
- `python 3.7.9`
- `numpy 1.19.4`
- `keras 2.3.0`
- `tensorflow 2.2.0`
- `scikit-learn 0.23.2`

Figure 2: Python Packages (for CNN Model)

- `nltk 3.5`
- `transformers 4.6.1`

Figure 3: Python Packages (for BERT Model)

3.2 Steps to follow

- **Step 1:** Create a conda environment and activate it using the command -

  ```
  conda activate <environment_name>
  ```

- **Step 2:** Using the appropriate version of your pip, verify all the packages listed above.

  ```
  pip list
  ```

- **Step 3:** In case required packages are missing, install the right version in your current conda environment using the following command-

  ```
  conda install <package_name>
  ```

Now, the other packages which has been used in the given code files, will be introduced in the following sections.

4 Understanding the Whole Collection

4.1 Raw Collection

In this section, the raw Data-Collection[1] has introduced. A detail data description has already been discussed in the main report (please refer the subsection 'Data Understanding' under the section of 'Methodology'). This collection consists of crawling news stories from 'Telegraph India' that were published from 2001 to 2010 over a ten-year span (see figure 4).

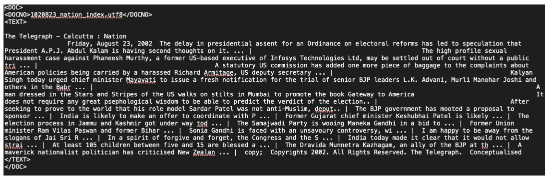
4.2 Structure of the Query (or Topic) Doc. & Relevant Judgement Doc (Ground Truth)

The dataset contains 25 plausible causal queries in total among which 20 randomly picked queries have been used in training the models and the rest of 5 queries have been used for testing purpose using 5-fold cross-validation i.e., the dataset's 25 causal queries will be split into five groups, and the model will go through five iterations of training and testing (Figure 5a)

Building a document pool for manual relevance evaluation in causal retrieval is more difficult than in conventional information retrieval (IR) for two key reasons. Since there

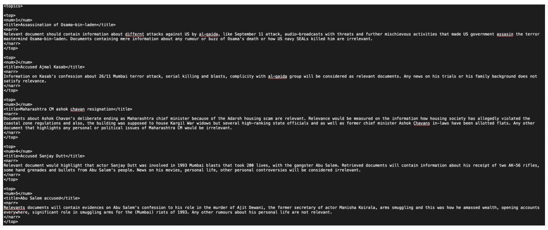
[1] to access the whole collection, please use this link `https://drive.google.com/file/d/1Mc0jLuXu9iccS41LIaGekSNc6WdEHzni/view?usp=sharing`

(a) Excerpts of the files

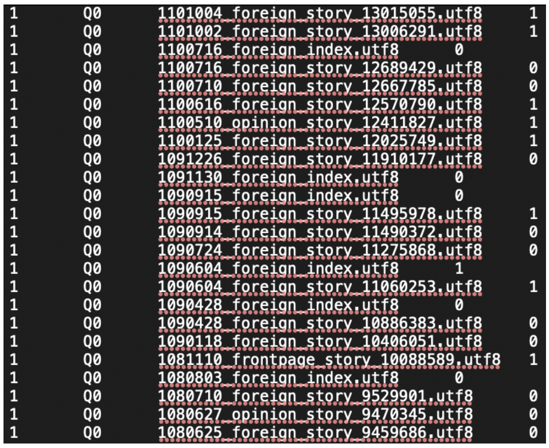


(b) Structure of each document

Figure 4: Structures of the raw collection



(a) Query doc.



```
1    Q0    1101004_foreign_story_13015055.utf8    1
1    Q0    1101002_foreign_story_13006291.utf8    1
1    Q0    1100716_foreign_index.utf8    0
1    Q0    1100716_foreign_story_12689429.utf8    0
1    Q0    1100710_foreign_story_12667785.utf8    0
1    Q0    1100616_foreign_story_12570790.utf8    1
1    Q0    1100510_opinion_story_12411827.utf8    1
1    Q0    1100125_foreign_story_12025749.utf8    1
1    Q0    1091226_foreign_story_11910177.utf8    0
1    Q0    1091130_foreign_index.utf8    0
1    Q0    1090915_foreign_index.utf8    0
1    Q0    1090915_foreign_story_11495978.utf8    1
1    Q0    1090914_foreign_story_11490372.utf8    0
1    Q0    1090724_foreign_story_11275868.utf8    0
1    Q0    1090604_foreign_index.utf8    1
1    Q0    1090604_foreign_story_11060253.utf8    1
1    Q0    1090428_foreign_index.utf8    0
1    Q0    1090428_foreign_story_10886383.utf8    0
1    Q0    1090118_foreign_story_10406051.utf8    0
1    Q0    1081110_frontpage_story_10088589.utf8    1
1    Q0    1080803_foreign_index.utf8    0
1    Q0    1080710_foreign_story_9529901.utf8    0
1    Q0    1080627_opinion_story_9470345.utf8    0
1    Q0    1080625_foreign_story_9459686.utf8    0
```

(b) Relevance Judgement doc

Figure 5: Structures of the Query & Relevance Judgement

is no recognized paradigm for causal IR, unlike standard IR, it might be difficult to incorporate pertinent information. In order to effectively analyze causal links, assessors also require prior knowledge of the event indicated in the inquiry. A multi-query formulation exploratory technique was employed to overcome this. During investigation, an interactive system assisted bookmark papers, highlighting those that could be relevant for developing causal relationships. These bookmarked documents were gathered into an evaluation pool together with the top 100 documents that were located using conventional IR models (e.g., LM, BM25, RLM). To establish the document's relevance, assessors made binary decisions based on their existing knowledge and the findings of their explorations (Figure 5b).

5 Data Pre-processing

5.1 Parsing the Raw Collection & Dumping in a File

- **Step 1:** In the very first step, the whole collection has been parsed using the XML parser. The detailed process about the .xml parsing has been given in the main report (please refer to the subsection 'Data Preparation' under the section 'Methodology').

- **Step 2:** Secondly, data cleaning has been performed. By data cleaning, we mean

```
public class MakeTelegraphDump {

   String             collectionPath;
   String             dumpPath;
   static int         docCount;
   static FileWriter    fileWriter;


   public MakeTelegraphDump(String collectionPath, String dumpPath) throws IOException {

      this.collectionPath = collectionPath;
      this.dumpPath = dumpPath;
      docCount = 0;
      fileWriter = new FileWriter(dumpPath + "telegraph_01_11.dump");
   }

   public void createDump(String collectionPath) throws FileNotFoundException, IOException {

      System.out.println("Dumping started...");
      File colFile = new File(collectionPath);
      if(colFile.isDirectory())
         collectionDirectory(colFile);
      else
         getFileContent(colFile);

   }

   private void collectionDirectory(File colDir) throws FileNotFoundException, IOException, NullPointerException {

      File[] files = colDir.listFiles();
      for (File file : files) {
         System.out.println("Dumping file : " + file);
         if (file.isDirectory()) {
            System.out.println("It has subdirectories...\n");
            collectionDirectory(file);  // calling this function recursively to access all the subfolders in the directory
         }
```

Figure 6: Making the Raw text collection

that all kinds of punctuation, special characters(i.e @, #, & etc.) and extra spaces has been removed.

- **Step 3:** Finally, we stem the raw texts to extract the root words and these analyzed texts are dumped in the file named 'telegraph_01_11.dump' (Figure 6)

5.2 Generating Vectors using Word2Vec

Next, all the raw texts that are dumped in the file 'telegraph_01_11.dump', we generate the 300 dimensional vectors using 'Word2vec' class available in 'gensim' library of python.

Note that for generating the vectors at the user's end, first save the whole data collection in the same directory and run the 'MakeTelegraphDump.java' script. The other option is to use the google drive link[2] given in the 'Description' file to access the vectors which are already uploaded for your convenience.

5.3 Indexing the Document using Lucene

Lucene is used to index the document files. Run 'Indexer.java' script to generate lucene index at your end. To get the more information about Lucene, please refer to the main

[2] https://drive.google.com/file/d/1bxYEdQBFBY56GqARb_xEobfbVS1ZXaEZ/view?usp=sharing

report file.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.StringReader;
import static java.lang.System.exit;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.TokenStream;
import org.apache.lucene.analysis.core.StopFilter;
import org.apache.lucene.analysis.en.EnglishAnalyzer;
import org.apache.lucene.analysis.tokenattributes.CharTermAttribute;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.index.IndexWriterConfig;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;


public class Indexer{

    String              collectionPath;
    String              indexPath;
    static IndexWriter  writer;             //org.apache.lucene.index.IndexWriter
    Analyzer            analyzer;           //org.apache.lucene.analysis.Analyzer; we use same analyzer for searching
    String              stopWordPath;
    List<String>        stopWordList;
    static int          docCount;


    public Indexer(String collectionPath, String indexPath) throws IOException {

        this.collectionPath = collectionPath;
        this.indexPath = indexPath;
```

Figure 7: Required Packages for Lucene Indexing

Please note that the lucene index is available here[3] if you want to use already indexed files. Figure7 shows the code snippet of the lucene indexing including the list of required packages.

5.4 Generating query-document interaction semantics

The next step is to generate query-document semantic similarities which we call interaction matrices. This module is written in java using the lucene index produced in the previous step. All the required libraries and codes are available in the folder 'Data_preprocessing → Interaction_matrix_generator'. Lucene libraries can be imported from the path 'Data_preprocessing → Interaction_matrix_generator → InteractionMatrix → lib'. For any given set of queries and collection index, run the bash script 'Data_preprocessing → Interaction_matrix_generator → InteractionMatrix → interaction.sh' to generate interaction matrices. A snippet of the bash script is given below in Figure 8.

6 Modelling

6.1 CNN-based Model

All the artifacts of the CNN-based proposed model can be found in the directory, 'CNN_model'. The code snippets of the required packages and CNN modelling are shown in Figure 9

[3] https://drive.google.com/file/d/1bxYEdQBFBY56GqARb_xEobfbVS1ZXaEZ/view?usp=sharing

```
#!/bin/bash
# generates interaction matrices between query and
# pseudo-relevant documents

if [ $# -le 8 ]
then
    echo "Usage: " $0 " <following arguments in the order>";
    echo "1. Query file (in .xml format).";
    echo "2. Path of the lucene index."
    echo "3. Stopwords file."
    echo "4. SimilarityFunction: 0.DefaultSimilarity, 1.BM25Similarity, 2.LMJelinekMercerSimilarity, 3.LMDirichletSimilarity.";
    echo "5. No. of top documents to retrieve.";
    echo "6. Path of the directory to store initial retrieved documents.";
    echo "7. Word vector file path.";
    echo "8. Name of the field used for searching (default 'content'- if using available index with this project).";
    echo "9. Interaction matrix path.";
    exit 1;
fi

queryPath=`readlink -f $1`
indexPath=`readlink -f $2`
stopFilePath=`readlink -f $3`
numHits=$5
retFilePath=`readlink -f $6`      # absolute directory path of the .res file
retFilePath=$retFilePath"/"
wordVecPath=`readlink -f $7`
searchField=$8
interMatrixPath=`readlink -f $9`
interMatrixPath=$interMatrixPath"/"

echo "Using query file at: "$queryPath
echo "Using index at : "$indexPath
echo "Using stop file at : "$stopFilePath
echo "Store initial retrieved file at :"$retFilePath
echo "Using word2vec file at : "$wordVecPath
echo "Store interaction matrices at :"$interMatrixPath

similarityFunction=$4

case $similarityFunction in
    1) param1=1.5
       param2=0.6 ;;
    2) param1=0.6
       param2=0.0 ;;
    3) param1=1000
       param2=0.0 ;;
esac

echo "similarity-function: "$similarityFunction" " $param1

# making the .properties file
cat > interaction.properties << EOL

queryPath=$queryPath

indexPath=$indexPath

stopFilePath=$stopFilePath

similarityFunction=$similarityFunction
interaction.sh
```

Figure 8: Bsash script for generating interaction matrices.

and 10, respectively.

6.2 BERT-based Model

Similar to the CNN model, we provide all the necessary artifacts in the directory, 'BERT_model'. The code excerpts of the required packages and BERT model are shown in the following figures 11 and 12, respectively.

```
import sys, os, random
import numpy as np
import keras
import pandas as pd
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, Input, Conv1D, Conv2D, MaxPooling1D, MaxPooling2D
from keras.layers.merge import concatenate
from tensorflow.keras import layers
import tensorflow as tf
from sklearn.metrics import accuracy_score


if len(sys.argv) < 4:
    print('Needs 3 arguments - \n'
          '1. Batch size during training\n'
          '2. Batch size during testing\n'
          '3. No. of epochs\n')
    exit(0)

seed_value = 12321
os.environ['PYTHONHASHSEED'] = str(seed_value)
random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value)
np.random.seed(seed_value)

# command line both for train and test
DATADIR_train = '/store/causalIR/train_hist/'   # (1)
DATADIR_test = '/store/causalIR/test_hist/'   # (1)

# A matrix is treated a grayscale image, i.e. am image with num_channels = 1
NUMCHANNELS = 1
# HIDDEN_LAYER_DIM = 16
# Num top docs (Default: 10)
K = 1
# M: bin-size (Default: 30)
M = 120 # depends on max query length
BATCH_SIZE_TRAIN = int(sys.argv[1])   # (7 - depends on the total no. of ret docs)
BATCH_SIZE_TEST = int(sys.argv[2])
EPOCHS = int(sys.argv[3])  # (8)


class InteractionData:
    # Interaction data of query qid with K top docs -
    # each row vector is a histogram of interaction data for a document

    def __init__(self, docid, dataPathBase=DATADIR_train):
        self.docid = docid
        histFile = "{}/{}.hist".format(dataPathBase, self.docid)
        # df = pd.read_csv(histFile, delim_whitespace=True, header=None)
        # self.matrix = df.to_numpy()
        histogram = np.genfromtxt(histFile, delimiter=" ")
        self.matrix = histogram[:, 4:]


class PairedInstance:
    def __init__(self, line):
        l = line.strip().split('\t')
        if len(l) > 2:
            self.doc_a = l[0]
causal_cnn_model.py
```

Figure 9: Required packages and input snippet.

```
def build_siamese(input_shape_top, input_shape_bottom):
    input_a_top = Input(shape=input_shape_top, dtype='float32')
    input_a_bottom = Input(shape=input_shape_bottom, dtype='float32')

    input_b_top = Input(shape=input_shape_top, dtype='float32')
    input_b_bottom = Input(shape=input_shape_bottom, dtype='float32')

    matrix_encoder_top = Sequential(name='sequence_1')
    matrix_encoder_top.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape_top))
    matrix_encoder_top.add(MaxPooling2D(padding='same'))
    matrix_encoder_top.add(Flatten())
    matrix_encoder_top.add(Dropout(0.2))
    matrix_encoder_top.add(Dense(128, activation='relu'))

    matrix_encoder_bottom = Sequential(name='sequence_2')
    matrix_encoder_bottom.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape_bottom))
    matrix_encoder_bottom.add(MaxPooling2D(padding='same'))
    matrix_encoder_bottom.add(Flatten())
    matrix_encoder_bottom.add(Dropout(0.2))
    matrix_encoder_bottom.add(Dense(128, activation='relu'))

    encoded_a_top = matrix_encoder_top(input_a_top)
    encoded_a_bottom = matrix_encoder_bottom(input_a_bottom)
    merged_vector_a = concatenate([encoded_a_top, encoded_a_bottom], axis=-1, name='concatenate_1')

    encoded_b_top = matrix_encoder_top(input_b_top)
    encoded_b_bottom = matrix_encoder_bottom(input_b_bottom)
    merged_vector_b = concatenate([encoded_b_top, encoded_b_bottom], axis=-1, name='concatenate_2')

    # ==============================

    merged_vector = concatenate([merged_vector_a, merged_vector_b], axis=-1, name='concatenate_final')
    # And add a logistic regression (2 class - sigmoid) on top
    # used for backpropagating from the (pred, true) labels
    predictions = Dense(1, activation='sigmoid')(merged_vector)

    siamese_net = Model([input_a_top, input_a_bottom, input_b_top, input_b_bottom], outputs=predictions)
    return siamese_net

siamese_model = build_siamese((K, M, 1))
siamese_model.compile(loss = keras.losses.BinaryCrossentropy(),
                      optimizer = keras.optimizers.Adam(),
                      metrics=['accuracy'])
siamese_model.summary()

training_generator = PairCmpDataGeneratorTrain(allPairsList_train, dataFolder=DATADIR_idf+'train_input/')
siamese_model.fit_generator(generator=training_generator,
                            use_multiprocessing=True,
                            epochs=EPOCHS,
                            workers=4)

# siamese_model.save_weights('/store/causalIR/foo.weights')
test_generator = PairCmpDataGeneratorTest(allPairsList_test, dataFolder=DATADIR_idf+'test_input/')
predictions = siamese_model.predict(test_generator)  # just to test, will rerank LM-scored docs
# print('predict ::: ', predictions)
# print('predict shape ::: ', predictions.shape)
with open(DATADIR + "12april.test.res", 'w') as outFile:     # (9)
    i = 0
    for entry in test_generator.paired_instances_ids:
:
```

Figure 10: Code snippet of the CNN model.

```
import pyterrier as pt
if not pt.started():
    pt.init()
from pyterrier_pisa import PisaIndex
import ir_datasets
import random
import torch
from transformers import BertModel, BertTokenizer, BertConfig
from transformers import logging
import itertools
import torch.nn.functional as F
import argparse
import more_itertools
import numpy as np
import os

torch.manual_seed(0)
logger = ir_datasets.log.easy()


def position_encoding_init(max_pos, emb_dim):
    position_enc = np.array([
        [pos / np.power(10000, 2 * (j // 2) / emb_dim) for j in range(emb_dim)]
        if pos != 0 else np.zeros(emb_dim) for pos in range(max_pos)])
    position_enc[1:, 0::2] = np.sin(position_enc[1:, 0::2])  # dim 2i
    position_enc[1:, 1::2] = np.cos(position_enc[1:, 1::2])  # dim 2i+1
    return torch.from_numpy(position_enc).type(torch.FloatTensor)


class BertQppModel(torch.nn.Module):
    def __init__(self, model_name='bert-base-uncased'):
        super().__init__()
        self.emb_dim = 768
        self.max_pos = 1000
        self.position_enc = torch.nn.Embedding(self.max_pos, self.emb_dim, padding_idx=0)
        self.bert = BertModel.from_pretrained(model_name)
        self.lstm = torch.nn.LSTM(input_size=self.emb_dim, hidden_size=self.bert.config.hidden_size,
                                  num_layers=1, bias=True, batch_first=False, dropout=0.2)
        self.utility = torch.nn.Sequential(
            torch.nn.Linear(self.bert.config.hidden_size, 100),
            torch.nn.Linear(100, 5),
            torch.nn.LogSoftmax(dim=1)
        )

    def forward(self, pos_list, input_ids, attention_mask, token_type_ids):
        res = self.bert(input_ids, attention_mask, token_type_ids).last_hidden_state  # [BATCH, LEN, DIM]
        res = res[:, 0]  # get CLS token rep [BATCH, DIM]
        res = res + self.position_enc(torch.tensor([pos for pos in pos_list], dtype=torch.long))  # [BATCH, DIM]
        res = res.unsqueeze(1)  # [BATCH, 1, DIM]
        lstm_output, recent_hidden = self.lstm(res)  # [BATCH, DIM]
        return self.utility(recent_hidden[0].squeeze(1))


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--index', default='/store/index/msmarco-passage.pisa')
    parser.add_argument('--batch-size', default=4, type=int)
    parser.add_argument('--train-its', default=250000, type=int)
    parser.add_argument('--chunk-per-query', default=25, type=int)
    parser.add_argument('--docs-per-query', default=1000, type=int)
bert_train.py
```

Figure 11: Required packages and input snippet of BERT model.

```
tokeniser = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertQppModel()
train_iter = _build_input(train_queries)
dev_iter = _build_input(dev_queries)
optim = torch.optim.Adam(model.parameters(), lr=2e-5)
suffixes = []
model_no = 0
u_loss = 0
count = 0
utl_loss = []
min_valid_loss = np.inf
if args.skip_utility:
    suffixes.append('noutil')
if args.skip_norel:
    suffixes.append('norel')
suffixes.append('{}')
model_name = f'../models/model-{"-".join(suffixes)}.pt'
with logger.pbar_raw(total=args.train_its, ncols=200) as pbar:
    # train the model
    for train_i in range(len(train_queries) * args.chunk_per_query * args.batch_size):
        query, docs, numrel, poshit = next(train_iter)
        count += 1
        inputs_train = tokeniser([query for _ in docs], [doc for doc in docs], padding=True,
                                 truncation='only_second',
                                 return_tensors='pt')
        utility = model(poshit, **{k: v for k, v in inputs_train.items()})
        # print('STEP LOSS : ', F.nll_loss(utility, torch.tensor([numrel])).item())
        u_loss += F.nll_loss(utility, torch.tensor([numrel])).item()
        if count == args.chunk_per_query:
            u_loss /= args.chunk_per_query
            u_loss = torch.tensor([u_loss], requires_grad=True)
            u_loss.backward()
            optim.step()
            optim.zero_grad()
            if u_loss.cpu().detach().item() != 0:
                utl_loss.append(u_loss.cpu().detach().item())
            u_loss = 0
            count = 0
            pbar.set_postfix(
                {'avg_utl_loss': sum(utl_loss) / len(utl_loss),
                 'recent_utl_loss': sum(utl_loss[-100:]) / len(utl_loss[-100:])})
            pbar.update(1)

        # validation after no. of iteration
        if (train_i+1) % 20000 == 0:
            # model_no = train_i
            valid_loss = 0.0
            model.eval()
            print('\n============ i am in validation ===========')
            for valid_i in range(len(dev_queries) * args.chunk_per_query * args.batch_size):
                query, docs, numrel, poshit = next(dev_iter)
                inputs_train = tokeniser([query for _ in docs], [doc for doc in docs], padding=True,
                                         truncation='only_second',
                                         return_tensors='pt')
                utility = model(poshit, **{k: v for k, v in inputs_train.items()})
                loss = F.nll_loss(utility, torch.tensor([numrel]))
                # Calculate total valid Loss
                valid_loss += loss.item()
            print(f'Training Loss: {sum(utl_loss) / len(utl_loss)} \t\t '
```

Figure 12: Code snippet of the BERT model.