**Geo Mapping of Entities Using BERT, Team 3**

**Sai Charan Akkena**

**Final Project Report**

**SECTION 1**

**Abstract**

The aim of this project is to build a model that enables researchers to make quantitative arguments about environmental awareness and cultural change. We are mapping a particular topic to a keyword which in our case is Wiki\_Words. We will be using an open-sourced neural network-based technique or NLP called BERT (Bidirectional Encoder Representations from Transformers). BERT is already pre-trained on the whole of Wikipedia and can be fine-tuned on smaller task-specific datasets.

**Data**

**1.1 Data Source**

The dataset has been taken from reddit and from scholarly articles which has the data from local news of the region for which we are analyzing the environmental changes. The data is in the format of compressed tar.gz file which has 118,059 files inside it in the format of .txt and has an actual size of 322MB .To ensure that the program doesn’t run out of memory, we will train our dataset in multiple batches.

**1.2 Data Format**

This data is comprised of .txt files which are in natural text language and processed with the Illinois Wikifier. The datasets are all in English except one which is in Norwegian. The data in the text files is comprised of various types of words and sentences, sentences with open and close parenthesis in the beginning and at the end, words in both wikified and non-wikified formats, text files with different sentences which are separated by a delimiter(.) and text files which do not have any delimiter(.) etc.

**1.3 Programming Language, packages**

The Project is implemented by using Python and its API’s. The Platform used in our project is Google Colaboratory, it allows us to write and execute arbitrary python code through the browser. Google Colab is a Jupyter Notebook environment that does not require any setup and runs entirely on cloud. Since the dataset is huge, we used our local machine’s GPU to test our code on few files and while running the code on the entire dataset, we switched to Google Cloud TPU. We used many machine learning libraries like PyTorch, Pandas, TensorFlow, Transformers by hugging face etc.

**Experiments**

**2.1 Preliminary Results**

As mentioned earlier, our data is in .txt files which contains different paragraphs with various sentences. Pre-processing of the dataset will perform stemming, tokenize words, remove punctuations and stop words, then the .txt files are merged into .csv format which is finally converted to .tsv format with columns understandable by BERT model. The data has been loaded into a pre-trained BERT Model and a fine-tuned model based on our Vocabulary has been built. Then the model is tested on the SQuAD dataset (Stanford Question Answering Dataset).

**SECTION 2**

**1. Data statistics**

Since our dataset has been extracted from reddit searching for the word ’humanities’, and it has already been pre-processed and wikified using the Illinois Wikifier, the wiki words are in the form of ‘wiki\_\_XXX\_\_<wikifier\_end\_parts>’. The statistics about WikiWords in the dataset are shown in Table 1, and the details of the top 10 frequent wiki words in the data set are provided in Table 2.

|  |  |
| --- | --- |
| **Description** | **Count** |
| Total number of wikiwords | 1,321,967 |
| Total number of wikiwords after removing duplicates | 78,598 |
| Single unique wikiwords after matching with BERT vocabulary | 3,718 |
| Multiple unique wikiwords after matching with BERT vocabulary | 14,039 |

Table 1: Description of the Dataset

**2. Experiments**

**2.1 Approach**

We used Hugging Face’s transformer for implementing our BERT model. It is a PyTorch implementation of Google’s BERT Model. We decided to use it due to the several advantages it offers us and also gives us best results which we aim for. It has the same accuracy rate as Google’s BERT implementation. There are many pretrained models shared by many people which will allow us to improve our accuracy and precision while not needing us to pre-train a newmodel. It has many functions already implemented which will make our task easy.

We will be extracting the wiki words, cleaning those words (i.e., removing Wikifier information at the beginning and at the end), find the frequency of each unique word and compare them with the existing ‘BERT vocabulary’ to find the matching words.

**2.2 Implementation**

This approach helps us while finding the frequency of the words after removing Wikifier Information. It will help us decide which words are important and which are not important to the BERT Vocabulary. For example, let us consider two different sets of words , one set with words that have frequency greater than 100 and the other with frequency less than 5. Out of these two sets we will know that words that occur more than 100 times will have more contexts for BERT to learn whereas words that occur less than 5 times don’t have more contexts for BERT to learn, which eventually results in BERT with less efficiency. As our project focuses on environmental awareness, it is necessary for BERT to analyze the text in the best way possible, for which data with a higher number of contexts will help.

We used hashlib package in the process of removing duplicate words after removing Wikifier Information. We used nltk package to find the frequency of the wiki words in the file and also remove the stop words. After comparing both the lists with the BERT vocabulary, we analyzed the matched words from lists.

For tokenization, since we are using Hugging Face’s transformers library, we will be using the provided Bert Tokenizer. It is based upon Google's Word Piece tokenizer and has traditional tokenization features such as lower casing which allows us to skip the pre-tokenization of the dataset. We assigned the maximum sequence length of 512 which is the longest length allowed by Bert base uncased model. We then convert tokens to the index numbers corresponding in the BERT vocabulary.

**2.3 Analysis**

Firstly, we counted the total number of words in the entire reddit dataset and extracted all the wiki words from the dataset and placed them in a separate text file (wiki\_word\_dataset.txt). We then have counted the total number of wiki words from the dataset. After this we have cleaned the wiki words, i.e. removing Wikifier Information at the beginning and end of each wiki word.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Wiki\_Words** | **arts** | **liberal** | **humanities** | **university** | **social** | **academic** | **school** | **term** | **united** | **states** |
| **Frequency** | 50,622 | 45,000 | 44,215 | 32,066 | 28,232 | 23,729 | 21,428 | 19,454 | 18,023 | 17,645 |

Table1.1:Top 10 Wiki\_Words in the dataset

After removing the Wikifier Information we will be having words that can be helpful to the model. But we cannot train BERT with all the words we got, as this might end up underfitting the model. To overcome this, we found the frequency of every word after removing Wikifier Information. The words ‘arts’ , ‘liberal’, ’humanities’, ’university’, ’social’ etc., are the most frequently occurring words in the dataset whereas the words 'damn', 'reichsbank' etc., have the least occurrence in the entire dataset. From the output of this task, we could see that occurrence of words is in a range from 1 to 50,000+.

Among these words, words that occur more than 50 times in this dataset will have a different type of contexts that will eventually help BERT in analyzing more about that word, whereas words with occurrence less than 30 to 50 times will considerably have different contexts, this can provide more information about the word and its usage, but the words which occurred less than 20 times in this huge dataset is not a big issue, as training BERT on these words will narrow the analyzing of BERT which results in less efficiency (i.e., the error rate may be higher because of these words).

We then matched these unique words file with the existing BERT vocabulary (which has word count more than 30,000) and wrote them in a new separate file, so that we can know the words that need to be added into the BERT vocabulary. We then made two lists from the unique words file, List 1 will contain all the single word after removing Wikifier Information and List 2 will contain all the other set of words. We then compared these two lists with BERT vocabulary and generated two separate files.

We had the choice of several libraries and APIs for tokenization, but we felt that it was best to use the BertTokenizer function in the HuggingFace Library itself. It saves us an extra step and also the time of doing the tokenization. After going through BertTokenizer documentation, we found out that it has all the features and functions that are present in other tokenizers such as nltk, spaCy and regular expression tokenizer.

Example of this tokenizer is ; [‘[CLS]’, ’the’, ’earth’, ’revolves’, ’around’, ’the’, ’sun’ ’[SEP]’].

A few challenges we faced are during cleaning of the wiki words, removing the wikifier Information. Since most of the words have Wikifier Information in the format wiki\_<word>\_<suffix> except some words which have two underscores between each separate word (example Wiki\_\_W\_\_E\_\_B\_\_Du\_Bois\_\_aqqaa).

While attempting to remove wikifier Information, we got ‘W’ only as a separate word and for certain cases there was three underscores after ‘Wiki’ while cleaning they were taken as empty lines, to solve this issue we used the replace() function and extracted the words.

**Section 3**

**3. Experiments:**

**3.1 Approach**

Previously, we had identified the wiki words, that are matching in the BERT Vocabulary after removing the Wikifier Information from the beginning and also at the end of the word, now for each document/wiki word pair we need to get the topic assignment. There are a couple of approaches for this:

1) One is for each occurrence of a wikiword in a document, we match the word with BERT vocabulary and if it is present we get the BERT embedding for the wikiword. We then use the dimension of the Word Embedding which has the highest absolute value as the topic assignment of that particular document.

2) Another approach is to get the Word embedding for the wikiword’s matching in BERT Vocabulary and cluster them together and use that particular cluster ID as the topic assignment for that document/wikiword pair.

We found the embedding vectors of the wiki words which are matching in BERT Vocabulary. Firstly, we extracted the sentence that contain the wiki\_words and then found the word embeddings of the wiki\_words. After extracting the embeddings, we found out the cosine similarity between the word embeddings to see how similar or different the words are from each other. Then we grouped the dataset using the index of maximum value of the embeddings vectors. We then finally performed K-Means clustering on the word embeddings.

Since there are many words which are present in BERT Vocabulary and also many words which are not present in BERT Vocabulary we took an approach where we take the wikiword’s that are matching with the Vocabulary and get their embeddings by using an encoding service called “Bert as a service ”. As required we then calculate the maximum value of particular wikiword embedding and get the index of that value and print it along with the name of the document, wikiword, topic assignment(index of the maximum value) in a text file.

Bert-as-Service is an online BERT encoder python service which makes it easier and less time consuming to find word embeddings. It is a stream-as-you-go service which allows us to extract embeddings in minimal time, using it we can compute similarity or relatedness between query and document, also cluster the words with the same topics together.

**3.2 Implementation**

As discussed above we first extracted the word embeddings using Bert-as-Service it basically creates a server which we can access using python code in our notebook/terminal, each time we send a word/sentence as a list, it will return the embeddings for the words/sentences. Each sentence/word is translated to a 768-dimensional vector. The Dimension of the embedding depends on the pretrained BERT Model we are using, here we are using “BERT-Base Uncased” which has 12-layers, 768-hidden units, 12-heads, 110M parameters , the dimension of the vector also depends on pooling\_strategy and pooling\_layer which can be set.

By default, Bert as a service works on the second last layer, i.e. pooling\_layer = -2. we can change it by setting pooling\_layer to other negative values, e.g. -1 corresponds to the last layer and -3 corresponds to the 10th layer. A pretrained BERT model has 12/24 layers, each “self-attends” on the previous one and outputs a [batch\_size(B), seq\_length(T), num\_hidden(D)] tensor. Here our goal is to get the word embedding, but if the goal is getting a sentence embedding, we need to pool the [B,T,D] tensor into a [B,D] embedding matrix. There are different pooling strategies such as average pooling, max pooling, hierarchical pooling, even concatenating avg-max pooling can be applied here directly as they do not introduce new parameters (so no extra training).

Different BERT layers capture different information It is said that consecutive layers have similar representation, whereas the first few layers represent considerably different meaning compared to the last few layers. The deepest layer is the one next to the word embedding. It may preserve the very original word information (with no self-attention etc.). On the other hand, we may achieve the very same performance by simply using word embedding. Therefore, using anything in-between the first layer and the last layer is a balanced approach.

In BERT two special tokens [CLS] and [SEP] are padded to the beginning and the end of an input sequence, respectively. Once fine-tuned with specific tasks, the embedding of those two tokens can represent the whole sequence, thus, we can include them in as well. However, if the BERT model is only pretrained and not fine-tuned on any specific task, embeddings on those two symbols are meaningless.

BERT is a model pretrained with two targets: masked language model and next sentence prediction. The last layer is trained in the way to fit this, making it too “biased” to those two targets. Here we can take the second-to-last layer.

There are 2 parts of Bert-as-service - BertServer and BertClient, first we start BertServer which will perform all the embedding functions. Starting Server is a Mandatory step. Next is BertClient(),BertClient() is a client object connected to BertServer, it sends the information to BertServer() that a particular ‘word’ needs to be encoded. BertServer() sends back the encoding to BertClient() which is then displayed to us. Next using scikit learn’s cosine similarity library we found out the cosine similarity between pairs of highest occurring single words in Bert. Cosine Similarity is calculated as dot product of X and Y which in our case are 2 tokens:

K(X,Y) = X.Y/||X||x||Y||

Now that we finally got the Word Embeddings, we now grouped the words on the basis of index of the maximum value of the embedding vector. We found out the max value using .max() and found the index of the maxvalue using .argmax(). We now the output file in the format of  **[file\_name(.txt), wiki\_word, topic\_assignment].**

Example:

**{**172244\_172244\_universitywire\_bodypluralhumanitiesorhleadpluralhumanities\_1998-01-01\_1998-12-31\_0\_17\_0.txt,humanities, 105 **}**

**3.3 Indexing**

While calculating the Embeddings and their topic assignments, we tried three different methods and got some interesting results. Since in bert-as-a-service, the default layer is last layer i.e. “pooling\_strategy = -2”, we found out the indexes using the default method, we also got the contextual word embeddings by extracting the sentences which contain the wiki words and we also tried by concatenating the last 4 layers by using “-pooling\_layer  -4 -3 -2 -1”.

**3.4 Analysis and Results**

The reason we decided to use Bert-as-service is very simple. We looked into many ways to get embeddings. Usually, extracting word embeddings is a tedious task. First we need to fetch models, fetch the hidden states, and then get the embeddings from the hidden states. While fetching the model is easy, the main difficulty lies in the step of hidden states. Bert uncased model has four dimensions - layer number(12 layers), batch number(1 for us), word/token(1 for us), hidden unit/feature(768 layers). Reducing all this to the required form of [batch, token, features] takes a considerable amount of time. Therefore, we use Bert-as-Service here to skip all this and directly get the embeddings required.

While finding the cosine similarity we got some interesting results. Firstly, Cosine Similarity values were not extreme. As in, we didn’t get any value less than 0 or greater than or equal to 1 which can be attributed to the fact that Google’s Bert model has been trained on the Whole of Wikipedia data which means that there will be somehow even a little bit of similarity between words.

Our finding was the impact of context on these embeddings.  For example, let’s take two highest occurring single words which were in Bert - ‘humanities’ and ‘professor’. When finding the cosine similarity of these words, we get a value of 0.6036, but when inserted along with some contextual text , let’s say “the professor teaches humanities and he studies humanities ” we get a cosine similarity of 0.4756 which is different.

|  |  |
| --- | --- |
| Topic\_ID | No. of Wiki\_Words |
| 205 | 2903 |
| 1114 | 177 |
| 346 | 116 |
| 1460 | 101 |
| 1109 | 81 |
| 909 | 49 |
| 849 | 96 |

In Method 1, we found out that there are a total of 3718 unique wiki words. We can see that majority of the words are in the index 205 as seen in Table 2, and figure1 which contains words such as airline, aircraft, airbus, bmw,Cadillac etc.

Table 2 : Method 1

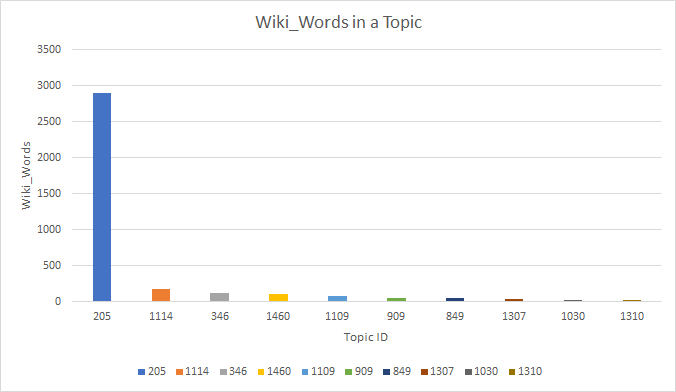


Figure 1

We ran similar tests on other words and found similar findings. It says that the “Word Embeddings” depend on the context of the sentence. Therefore, we feel that grouping on the basis of cosine similarity of the Wiki\_Words might not be the best approach.

|  |  |
| --- | --- |
| Topic\_ID | No. of Wiki\_Words |
| 2509 | 3655 |
| 6452 | 61 |
| 4181 | 2 |

In Method 2, we got total of 3718 unique value which are broken down into 3 indexes. When we concatenate 4 layers into one then the vector length is 4x768 = 3072, each vector has elements:

4x3072= 12,228.

Table 2.1: Method 2

Here we can find that the concatenation of the last four layers does not produce ideal grouping at all as everything is being grouped into only 3 indexes as seen in Table 2.

In the 3rd method, we found out the sentence embedding, and we got completely different results as seen in figure 2 and also in Table 2.3. When we get embeddings from context., we can see that we go different values and also the words are spread in different indexes.

We can also see that the same word here can have different embedding values which shows us how context plays a major role in influencing the embedding vectors.

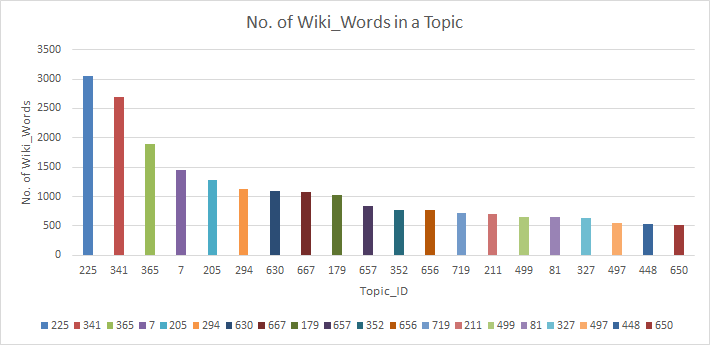
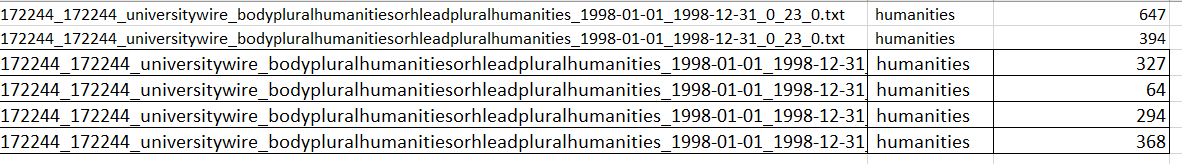


Figure 2

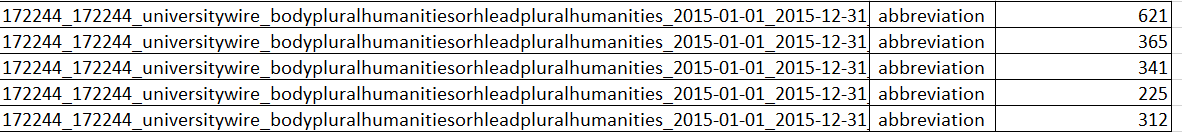
|  |  |
| --- | --- |
| Topic\_ID | No. of Wiki\_Words |
| 225 | 3060 |
| 341 | 2886 |
| 365 | 1897 |
| 7 | 1453 |
| 205 | 1286 |
| 294 | 1128 |
| 630 | 1089 |
| 667 | 1070 |

Table 2.3 – Method 3

As we can see in figure’s 3 and 4 the word’s humanities and abbreviation occur under different topic assignments, which shows that the words have different meaning when extracting the embeddings



‘humanities’ has different embeddings here which tells us that humanities has different meanings when extracting the vectors.



Same observation is being observed in ‘abbreviation’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Coherent Clusters** | | | | |
| **Topic ID** | **165** | **597** | **702** | **122** |
| **Sample wikiwords in that group** | april | algebra | australia | algebra |
| february | biochemistry | california | encyclopedia |
| january | biology | mexico | essay |
| march | mathematics | paris | homework |
| november | physics |  | librarian |
| october |  |  | professor |
| september |  |  |  |

**Table 8**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Incoherent Clusters** | | | | |
| **Topic ID** | **322** | **314** | **646** | **416** |
| **Sample wikiwords in that group** | ballet | adult | capitalism | aesthetics |
| inch | christmas | humanities | africa |
| lol | facebook | irony | border |
| mathematics | income | laundry | feminism |
| october | laptop | nonsense | happiness |
| theology | privacy | villain | probability |
|  | tuition |  | wisdom |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tasks** | **Task Description** | **Student 1 name** | **Student 2 name** | **Student 3 name** | **Student 4 name** |
| Task 1 | Extracting the Wiki Words | Suriya Prakaash | **Sai Charan** |  |  |
| Task 2 | Cleaning the extracted wikiwords |  | **Sai Charan** |  | Sai Vishal |
| Task 3 | Finding frequency of the wikiwords | Suriya Prakaash |  | Siddhant Jain |  |
| Task 4 | Removal of duplicates and Comparison with Bert vocabulary |  |  | Siddhant Jain | Sai Vishal |
| Task 5 | Separating Single and Multiple wikiwords | Suriya Prakaash |  |  | Sai Vishal |
| Task 6 | Word embedding extraction and grouping wikiwords using index of highest vector values |  |  | Siddhant Jain | Sai Vishal |
| Task 7 | Cosine Similarity | Suriya Prakaash | **Sai Charan** |  |  |
| Task 8 | Extracting context of single wiki words |  | **Sai Charan** | Siddhant Jain |  |
| Task 9 | Extraction of embedding vectors for single wikiwords with context and grouping using index of highest vector values | Suriya Prakaash | **Sai Charan** |  |  |
| Task 10 | Analysis of groupings formed in approach 1 |  |  | Siddhant Jain | Sai Vishal |
| Task 11 | Implementing K-means by using embedding vectors of single wikiwords with context | Suriya Prakaash |  |  | Sai Vishal |
| Task 12 | Analysis of groupings formed in approach 2 |  | **Sai Charan** | Siddhant Jain |  |