BERT for Geo Mapping Entities, Team 3
Suriya Prakaash Sundaram Kasi Thirunavukkarasu
Final Report

SECTION 1

ABSTRACT

The main objective of the project is to discover the topics or clusters of the Wikiwords. The dataset is extracted from user comments of 'reddit' based on the search with the word 'humanities'. Wikiwords are the entities that are detected in the dataset (for example, scientific articles discussing some geographic locations or institutions or organizations or etc.) Discovering the topics of the wikiwords is implemented using two different approaches, grouping using the index of the highest word embedding vector and clustering using the K-means algorithm with the word embedding vector values. First step in the process is to pre-process the dataset and extract the wikiwords and clean them (removing wikifier parts in those words). Next step is extracting the single wikiwords (for example, Wiki__Journalism__isqza) matching with BERT vocab. The last step is discovering topics of those wikiwords with the context they occur and do analysis on those clusters formed to verify whether the formed groups are good and correct. With the discovered topics or clusters of Wikiwords, extraction of geolocations of the wikiwords in a topic and map those geolocations to the wikiwords in that topic using the visualization tool.

1. Data

1.1 Data Source

The dataset is from reddit, which is a collection of articles on the search with the keyword 'humanities'. Dataset totally contains 118,059 files of size 484 MB on disk.

1.2 Data Format

The data consists of large number of .txt files which are completely in natural text language. The data is firstly processed with Illinois Wikifier. Almost all the data are in English with only one text file in Norwegian. Data in these text files are in the form of as sentences, sentences without delimiter, wikified words in those sentences.

1.3 Programming Language, packages

Major part of the project will be done using python and with many python APIs. To perform preprocessing of the data, we used python. As the dataset of this project contains files more than 100000, GPU is required because of which we used Google Colab (which runs completely on cloud) and certain computations were done using Jupyter notebook.

Libraries : sklearn, bert_serving, and collections.

Packages : numpy, sys, tensorflow, cosine_similarity, pandas, os, Counter, matplotlib, seaborn, sklearn, hashlib, nltk, spacy, BertClient, and MinMaxScaler.

2. Preliminary Experiments

2.1 Preliminary Results

Pre-processed dataset had the data in the following format,

- Wikifier tokens removed from the wikiwords,
- Two separated lists (List 1 – wikiwords that holds only one token, List 2 - wikiwords that holds more than one tokens),
- Frequency of each wikiwords in both lists,
- Removed duplicates in both lists,
- Extracted the wikiwords that matches with BERT vocab from both the lists,

- Punctuations removed, and
- Stop words removed.

SECTION 2

1 Data statistics

The statistics about wiki words in the dataset are as given in the **Table 1**. Details on the top 5 five most occurring wikiwords are given in the **Table 2.** The final analysis on these wiki words helped us to find that the words 'arts', 'liberal', 'humanities', 'university', 'social' etc., are the most frequently occurring words in the dataset. Whereas the words 'overy', 'shovelhead', 'goatherd', 'damn', 'flagstone' etc., have the least occurrence in the entire dataset.

Description	Count
Total number of wikiwords	1,321,967
Number of Wikiwords after removing duplicates	78,598
Single unique wikiwords matching BERT vocab	3,718
Multiple unique wikiwords matching BERT vocab	14,039

Table 1

Top 5 Wiki Words	arts	liberal	humanities	university	Social
Frequency	50,622	45,000	44,215	32,066	28,232

Table 2

2. Experiments

2.1 Approach

- To form groups among the wikiwords, we used two approaches as below
  1. Grouping using the index of the highest word embedding vector.
  2. Clustering topics based on their word embedding values using K-means model.
- We grouped single wikiwords with their context using Bert-as-a-service embedding vectors.
- We have chosen to implement the above two approaches on single wiki words and formed groups.
- We could see from the **Table 1** that total number of single unique wiki words matching with BERT vocab is 3,718.
- The occurrence rate or wikiwords in the dataset are as follows
  - more than 10,000 times are 21
  - occurred in between 50 to 10,000 times are 4,891
  - occurred in between 10 to 50 times are 8,255
  - less than 10 are 31,641
- We could see that the occurrence rate between words increases exponentially.
- As we chose to work on single wikiwords, focusing on wikiwords with occurrence rate than 10 was challenging, though the occurrence rate is less than 10 the number of words in that range is too high which cannot be ignored while forming groups.

bert-as-a-service:

Bert-as-service uses BERT as a sentence encoder and hosts it as a service via ZeroMQ, which is an open source messaging library, it allows us to map words or sentences into fixed-

length representations, it basically creates a server which we can access using python code in our jupyter notebook or terminal. Every time when we send a word or a sentence as a list, it will return the word embeddings for that in the form of vectors. It will start a service with given number of workers (in our case it is 6), meaning that it can handle up to six concurrent requests. It will return a ndarray, in which each row is the fixed representation of a word or a sentence. We can also get encodes of a pair of sentences by concatenating them using '|||',

e.g. bc.encode(['First do it ||| then do it right'])

Each sentence is translated to a 768-dimensional vector. The dimensions of the embeddings depends on the pretrained BERT model we are using, here we used 'BERT-Base Uncased' which has 12-layers, 768-hidden units, 12-heads, 110M parameters, the dimension of the vectors 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-attend 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 suppose if our goal is getting a sentence embedding, we need to pool the [B,T,D] tensor into a [B,D] embedding matrix using different pooling strategies such as average pooling, max pooling, hierarchical pooling etc.

BERT is a model pre trained 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 are using the second-to-last layer(11th).

There are 2 parts of Bert-as-service, such as

- BertServer, and
- BertClient

First, we will start BertServer which will perform all the embedding functions. Starting server is a mandatory step. Next is BertClient() which is a client object connected to BertServer, it sends the information to BertServer() that a particular 'word/sentence' needs to be encoded. BertServer() sends back the encoding to BertClient() which is then displayed to us.

Advantages:

- State of the art – build on 12/24 – layer BERT models.
- Easy to use – getting word embeddings is easy.
- Processing time is so fast, low latency.

Scalability – scale nicely and smoothly on multiple GPU and multiple clients without worrying about concurrency.

2.2 Describe what implementation you are using. How you are evaluating the quality of your results.

- We used hashlib package in the process of removing duplicate words after removing wikifier add-ons.
- We then used nltk package to find the frequency of the wiki words in the file.
- For the pre-processing part of the project, we used file concepts in python.
- In both the approaches we used BERT-as-a-service to obtain the word embedding vectors.
- For first approach, we used numpy, sys, tensorflow, BertClient, and pandas packages in python.

- From the results of our first approach, some of the topics discovered are completely coherent, some of the topics are partially coherent and some topics are incoherent.
- For second approach, we used numpy, sys, tensorflow, cosine_similarity, BertClient, pandas, Counter, matplotib, seaborn, sklearn, and MinMaxScaler packages.
- From the results of our second approach, we could see that some wikiwords grouped together are completely related one topic and most of the groups have few wikiwords that are partially related to that topic.

2.3 Give details about the approach you selected, present a summary of the test results, and provide brief analysis

- The first step in our process is pre-processing the wikiwords before forming groups on them.
- We have several tasks to get the data which will help BERT in-text analyzing efficiently. Firstly, we counted the total number of words in the entire reddit dataset.
- Next, I and Sai Charan extracted all the wiki words from the dataset and placed them in a separate text file (wiki_word_dataset.txt). Then I and Sai Charan have counted the total number of wiki words from the dataset.
- After this our team have cleaned the wiki words, that is we removed wikifier add-ons at the beginning and end of each wiki word. After removing the wikifier add-ons we have words for which topics must be discovered.
- Then I and Siddhant found the frequency of every word after removing wikifier add-ons. From the output of this task, we could see that occurrence of words is the range from 1 to 50,000+.
- Among these words, words that occur more than 10 times in this dataset will have a different type of contexts that might eventually fall into different topics.
- Whereas words with occurrence less than 10 times in such huge dataset is not a big deal, but still they might also have considerably different contexts, this can provide more information about the word's topic which is also needed for grouping.
- After this step, our team have removed the duplicates from the cleaned words file. 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 can be grouped together.
- I and Sai Vishal then made two lists from the unique words file, List 1 that contains all the single word after removing wikifier add-ons and List 2 that contains all the other set of words. We then compared these two lists with BERT vocabulary and generated two separate files.
- Some of the challenges we faced in these tasks are while cleaning the wiki words, removing the wikifier add-ons. As most of words have wikifier add-ons in the format 'Wiki__<word>__<wikifier_suffix>', some of the other different format of wikiwords available in our dataset are given in **Table 3.**

Different types of wiki words	Type
Wiki__Journalism__isqza	Single word
Wiki__Mass_communication__ibosles	Multiple words
Wiki__W__E__B__Du_Bois__aqqaa	Multiple words with Double underscore in between
Wiki____st_century__efslz	Words with more than double undercore
Wiki_____efqis	Just wikifier tokens

Table 3

- We then used Bert-as-a-service to get the word embeddings of the context that the wikiwords has occurred in both of our approaches.
- We have implemented two different approaches in forming groups among the wikiwords.
  1. Grouping wikiwords using the index of the highest embedding vector.
  2. Clustering topics based on their word embedding values using K-means model.
- After pre-processing is done, we had a separate file that holds all the single wikiwords. Then we extracted the context (totally 10 tokens including the cleaned wikiword) of the wikiword for which we found the embedding vectors. The extracted contexts are then stored in a file for further process.

SECTION 3

3. Experiments/ Results

3.1 Provide details on your work in the last 2 weeks. Focus on the results – provide good summaries of your results. Talk about parameter tuning if it is required for your approach.

After extracting the cleaned single matching wikiwords, we then proceeded to get the word embedding vectors for those words and group them using index of the highest vector value or by using k-means. Our results after this step was not making much sense as discovering topic of a wikiword alone is not enough.

Cosine similarity:

As we know in English same words can occur have different meanings, meaning of a word is always respective to the context the word has occurred. To identify the word's topic with respect to context, we used cosine similarity. I and Sai charan have tried various use cases for same word.

- In cosine similarity values, we know that if
  - the values are closer to 1, the words are highly similar
  - the values are closer to 0, then the words are not similar and
  - If the values are closer to -1, then the words are opposite.
- For example, we found cosine similarity two of the highest occurring single wiki words – 'humanities' and 'professor' the cosine similarity value is

  cosine_similarity value = 0.64957803
- But when we tried these words with a context, we get different values – 'the professor teaches humanities' and 'he studies humanities' the cosine similarity value is

  cosine_similarity value = 0.745095

The difference in the above two values are because of the other words which are added to the context. This addition of context to a word can increase or decrease the values significantly.

After considering this change in cosine similarity values with respect to the context, we formed groups by using the entire context or a sub part of the context in which the wikiword has

occurred, this will give us the topic that the wikiword would fall into, as context has a huge impact in identifying the topic of that wikiword.

3.2 Analyze your results. What approach/parameters are working well, what are not working well? Explain why you think it is the case

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(data pre-processing)		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(by using default method) and grouping wikiwords using index of highest vector values(Method 1)		Sai Charan	Siddhant Jain	
Task 7	Cosine Similarity	Suriya Prakaash	Sai Charan		
Task 8	Extracting embeddings (by concatenating last 4 layers) of wiki words(Method 2)		Sai Charan	Siddhant Jain	
Task 9	Analysis of groupings formed in Method 1			Siddhant Jain	Sai Vishal
Task 10	Extraction of embedding vectors from sentences for single wikiwords with context and grouping using index of highest vector values(Method 3)	Suriya Prakaash			Sai Vishal
Task 11	Analysis of groupings formed in Method's 2 and 3		Sai Charan	Siddhant Jain	
Task 12	Implementing K-means by using embedding vectors of single wikiwords with context	Suriya Prakaash			Sai Vishal

Experiment/Approach

Approach 1 – Grouping using index

In this approach,

- We tried this approach in 3 different ways to get the best groupings. In each way, we tested with different pooling strategy, pooling layer, and the input which we passed.
- The 3 different test runs are as following table

Test Run	Input	pooling_strategy	pooling_layer
1	Single wikiword	NONE	-2
2	Single wikiword	NONE	-4, -3, -2, -1
3	Context of single wikiword	Default	-2

- We set the pooling_strategy to NONE as it was the optimal way to get the word embeddings as described in the Bert-as-a-service documents.
- Generally, if the vector dimension is (1,3,768), then
  - 1 is number of sentences,
  - 3 is the number of tokens in that sentence, and
  - 768 features of the model
  - This means that the vector length is 2304

Analysis and results of 1st test run

- The results which we got were interesting.
- We removed the duplicates and obtained the unique number of words represented by the topic IDs.
- We found out that there are 3718 unique words. The top 7 topics with the most unique word counts are given in **Table 4.**
- **Chart 1** gives the top 10 topics with the most unique word counts.
- Topic ID – 205 contains most of the unique words. Some of the wikiwords in Topic ID - 205 are such as airline, aircraft, airbus, bmw, cadillac, etc.

Topic ID	205	1114	346	1460	1109	909	849
Word Count	2903	177	116	101	81	49	96

Table 5

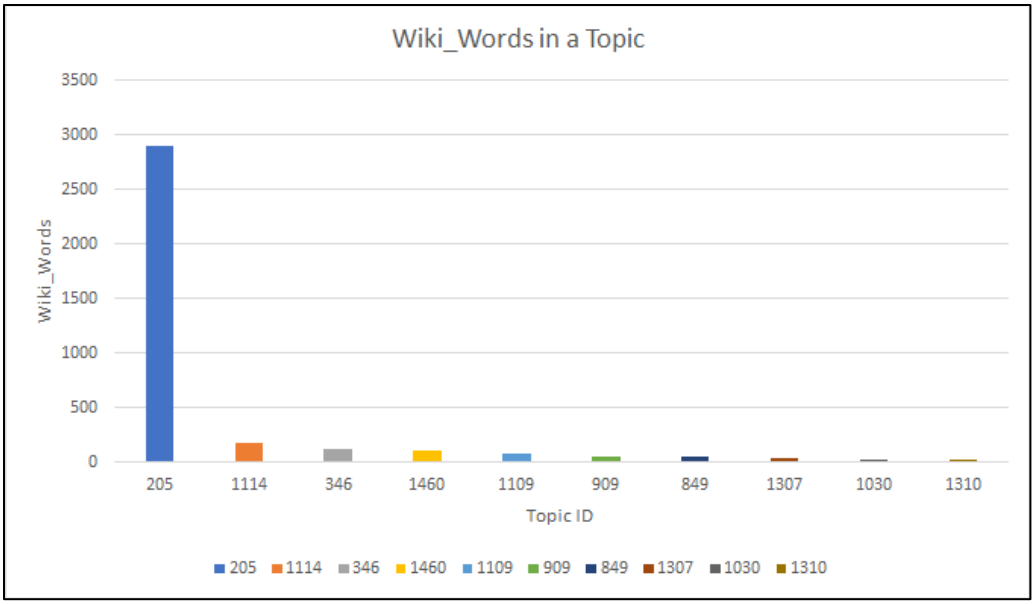

Chart 1

Analysis and results of 2nd test run

- In this test run, we concatenated 4 layers into 1 layer, then the vector length is 4x768 = 3072
- Each vector has elements 4x3072 = 12,228
- From the results of this test run we find that concatenation of the last four layers does not produce best groupings of single wikiwords of our dataset.
- As all the unique wikiwords are grouped under 3 topics as given in **Table 6**, which is in so generalized form.

Topic ID	Count
2509	3655
6452	61
4181	2

Table 6

Analysis and results of 3rd test run

- In this test run we gave input as sentences that has the single wikiwords in that.
- We passed every row in the file that holds the wikiwords context into BertClient.encode(), which will send the context to BertServer.
- In this test run, embedding vectors of dimension (1,768) for that sentence.
- I and Sai Charan then found the index of the highest embedding vector for each wikiword context.
- This index is the assigned topic ID for that wikiword in that context.
- Also, we can see that the same wikiword can have different topic ID which shows us how context influences the embedding values.
- This is final selected approach as the groupings formed are comparatively better than other two test runs.

- The top 8 topics with most unique word counts are given in **Table 7**.
- **Chart 2** gives the top 20 topics with the most unique word counts.
- Most of the coherent clusters have very few wikiwords that are irrelevant to that topic and incoherent clusters have most of the wikiwords in the group like each other.
- Some of the coherent and incoherent groupings are given in **Table 8** and **Table 9** respectively.

Topic ID	225	341	365	7	205	294	630	667
Count	3060	2886	1897	1453	1286	1128	1089	1070

Table 7

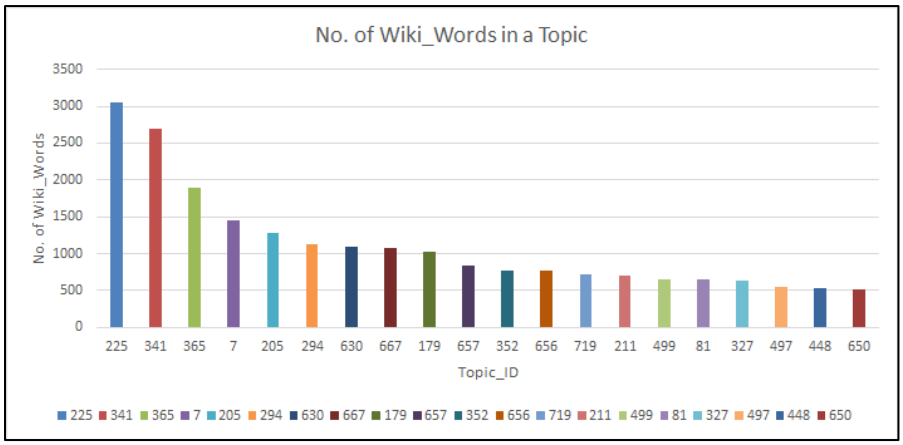

Chart 2

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

Table 9

- The final output file for this approach is delivered in the below format

- <document_name>, <wiki_word_after_removing_wikifier_tokens>, <topic_ID>
- For example,
  - 172244_172244_universitywire_bodypluralhumanitiesorhleadpluralhumanities_1998-01-01_1998-12-31_0_17_0.txt, humanities, 81

Approach 2 – Clustering wikiwords through K-means

- In our second approach, we tried clustering single wikiwords with their context using k-means model.
- In approach 1, we just used index of highest embedding vector value, but for this approach we will passing the entire dimension of vector value as input (i.e., 768 vector values).
- This dataset will require a greater number of clusters to get the best groupings, which can be found by using elbow method.
- But due to time constraints, we formed 'k = 200' clusters.

Analysis and results of the approach

- From the results of this approach, we could obviously see that grouping 350,000+ records into 200 clusters will combine more than one topic in a group.
- Most of the clusters are coherent, but the thing that cluster does have words related to another topic group. Some of the examples of such clusters are shown in **Table 10.**
- Whereas there are clusters which contains words that would not fall into any topic but grouped together based on the beginning character of the word. Some of the examples of such clusters are shown in **Table 11.**

Coherent clusters				
Cluster ID	**164**	**58**	**86**	**138**
Sample wikiwords in that group	scholarship	censorship	march	cancer
	humanities	narrative	september	human
	language	judaism	october	obesity
	seminar	socialism	november	hiv
	website	psych	humanities	fraternity
	curriculum	apprenticeship	auditorium	arab
	citizenship	biology	alumnus	morale
	mathematics		professor	islam

Table 10

Incoherent Clusters				
Cluster ID	**24**	**108**	**77**	**62**
Sample wikiwords in that group	accountability	hamburger	bog	laptop
	accountant	happiness	undercover	seconds
	activism	hawaii	nomad	adult
	actor	hectare	mantra	pumpkin
	addiction	hem	catering	interstate
	adjective	herb	geek	ink
	adolescence	heroin	interest	cough
	adult	hindu	monopoly	axle

Table 11

Drawbacks:

- The reason behind the noise in most of the clusters are because, grouping records of such greater size (368375 records) under 200 topics.
- Various wikiwords have occurred in more than one cluster.
- Average word count of each cluster (with duplicates) is 1842, and which is very high, which denotes that not all the wikiwords in the cluster is of same topic.
- The top 8 clusters with most unique word counts are given in **Table 12.**
- The top 8 clusters that has most word counts with duplicates are given in **Table 13.**
- **Chart 3** gives the top 17 clusters with the most unique word counts.
- **Chart 4** gives the top 12 clusters with the most unique word counts.

Cluster ID	77	80	117	112	158	62	87	157
Word count	1208	1052	1033	983	977	963	962	943

Table 12

Cluster ID	2	77	95	131	25	113	158	80
Word count	8522	3630	3484	3356	3321	3211	3185	3183

Table 13

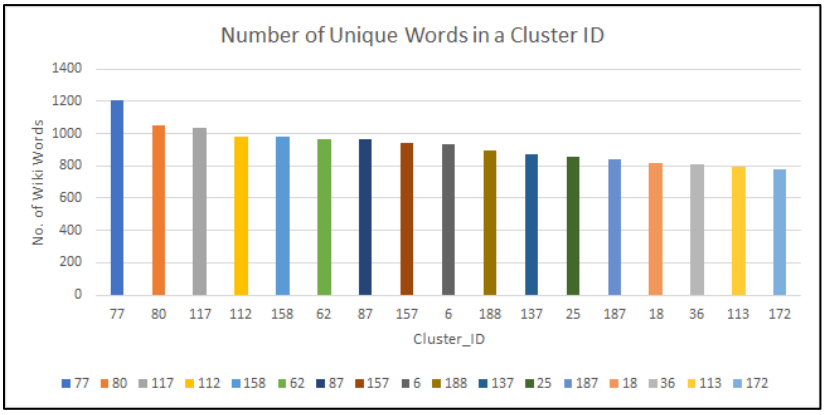

Chart 3

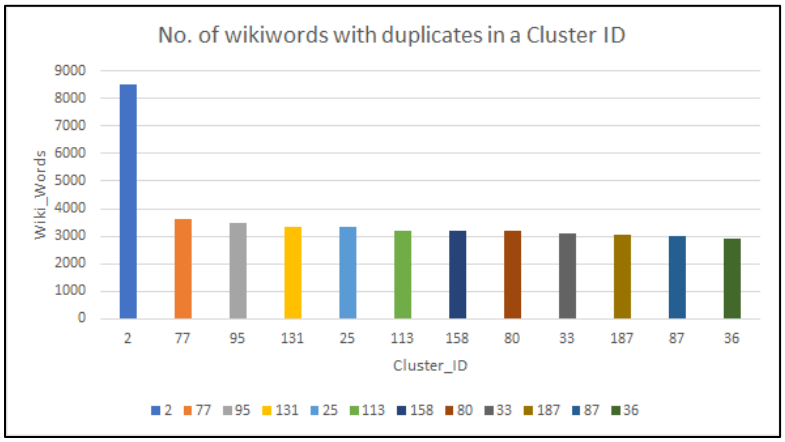

Chart 4

- The final output file for this approach is delivered in the below format
    - <document_name>, <wiki_word_after_removing_wikifier_tokens>, <cluster_ID>
- For example,
    - 172244_172244_universitywire_bodypluralhumanitiesorhleadpluralhumanities_1998-01-01_1998-12-31_0_17_0.txt, humanities, 22

3.3 Analyze the errors. If something did not work well, analyze why. For example, the approach may not be good for the data, in this case explain what characteristics of the data are causing it.

- Some of the errors which we faced in grouping the data are, grouping wikiwords using the word embedding vectors of that wikiword alone.
- We faced this issue in our test runs 1 and 2 of our approach 1, as this will have the same wikiwords in same group irrespective of the context.
- For example,

    'After stealing money from the bank vault, the bank robber was seen fishing on the Mississippi river bank'
- The word 'bank' in the above appeared 3 times, two of those are in same context and the remaining one occurrence is in different context, but by our 1st and 2nd test runs of approach 1, topic of all the three occurrence of 'bank' will be same and grouped together under same topic ID.
- The reason behind this error is we were grouping the wikiword using the embedding vector of that word alone, but this does not give the proper topic the word has to be grouped into.
- So, this approach is not good for our data.
- When tried in this approach we found that most of wikiwords (with duplicates) occurred in the dataset, grouped into same topic IDs.
- **Table 14** will give us the top 5 topic IDs and their word counts.

Topic ID	205	1109	1114	1460	849
Word count	189666	52502	16247	13542	11760

Table 14

- We could see that Topic '205' itself has more than half of the single wikiwords in the dataset which does not make sense.

Conclusion

- While BERT might not be ideal for this kind of grouping and clustering, we still got some surprising results which were fascinating and interesting.
- We feel that we could have got much better results if not for our time-constraint and lack of knowledge about functionalities of BERT in the earlier stage of the project.
- Indexing and clustering obtained are partially suited outcomes for this dataset.
- We got much better results with context but there is still room for improvement.
- Choosing which layer and pooling strategy to use influences our outcomes even more.
- While we are not getting results like LDA, we still got some unexpected gains.