

# ENTITY LINKING USING BERT SIMILARITY

**CSE2003 Data Structures and Algorithms**

**Project Report**

**Submitted by**

*A.V.N.M.HEMATEJA 19BEC1025*

*K.ESWAR GOWTHAM 19BEC1135*



**VIT<sup>®</sup>**  
**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)

**School of Computer Science and Engineering**

**Nov 2020**

## Table of Content

	Page Number
Introduction	- 3
Justification for DSA project	- 4 - 21
J component contribution (highlights)	- 22 - 23
Annexure (screenshot if any)	- 24

# ENTITY LINKING USING BERT SIMILARITY

## Introduction:

The main problem in NLP now a days is Entity Linking. Entity Linking is the process of finding the main entity to which given input text is connected to. There are so many methods for solving this problem. But the most interesting method is ABACO (Annotation BAsed on COherence). This method has the ability to become the best method in this field.

But, There are some drawbacks in ABACO. So in this project we are going to propose our changes to this method. Which will increase the accuracy in predicting the Entity.

For increasing Accuracy we are using BERT (Bidirectional Encoder Representations from Transformers). BERT is the real break through for NLP in the last year. This BERT has the ability to learn the meaning and context of the language. Which is the main property NLP needs. We are using this BERT for giving the Score between the wikipedia page and mention.

ABACO used just BM25 ranking. We are adding this BERT-similarity for creating the absolute increase in the accuracy. We also changed many other things in the process of ABACO for improving the outputs.

## JUSTIFICATION OF DSA - PROJECT:

We are proposing an Algorithm by changing the processes in ABACO.

The main 5 steps are:

1. Mention Detection.
2. Search in DBpedia.
3. Candidate set Generation.
4. Disambiguation.
5. Subgraph generation.

We will be starting the process by tokenizing the given text and we will be passing them through the N-gram Algorithm for making the N-grams which will be passed through POS tagger that tags every N-gram with parts of speech.

Then We are going to use Stanford - NER for detecting the Named Entity from tagged N-grams. After that we will be searching these in DBpedia using Label and Synonym indexes.

Label index:

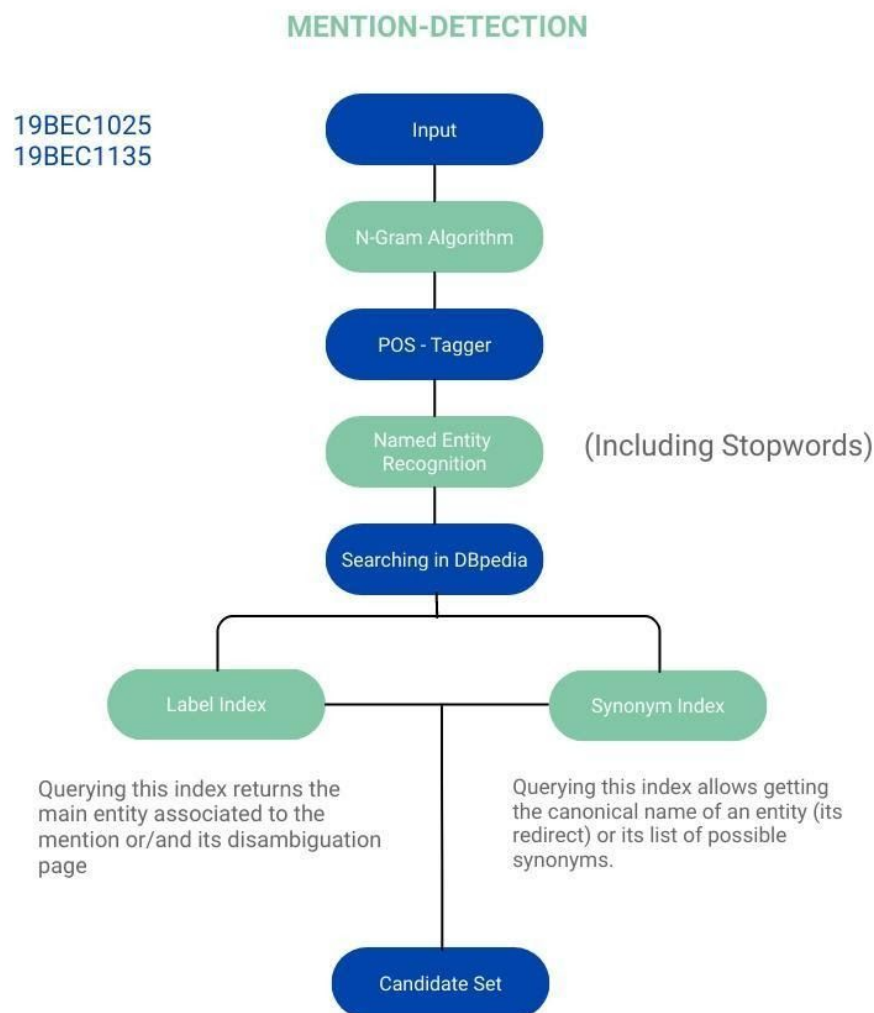
This index gives us the main entity related to the mention and also the disambiguation page for the mention.

Example: If we search for “dhoni” using Label index we will be getting main-entity : “MS Dhoni” and also disambiguation pages like “MSD”.

Synonym index:

This index gives us the possible synonyms or redirects for the given mention.

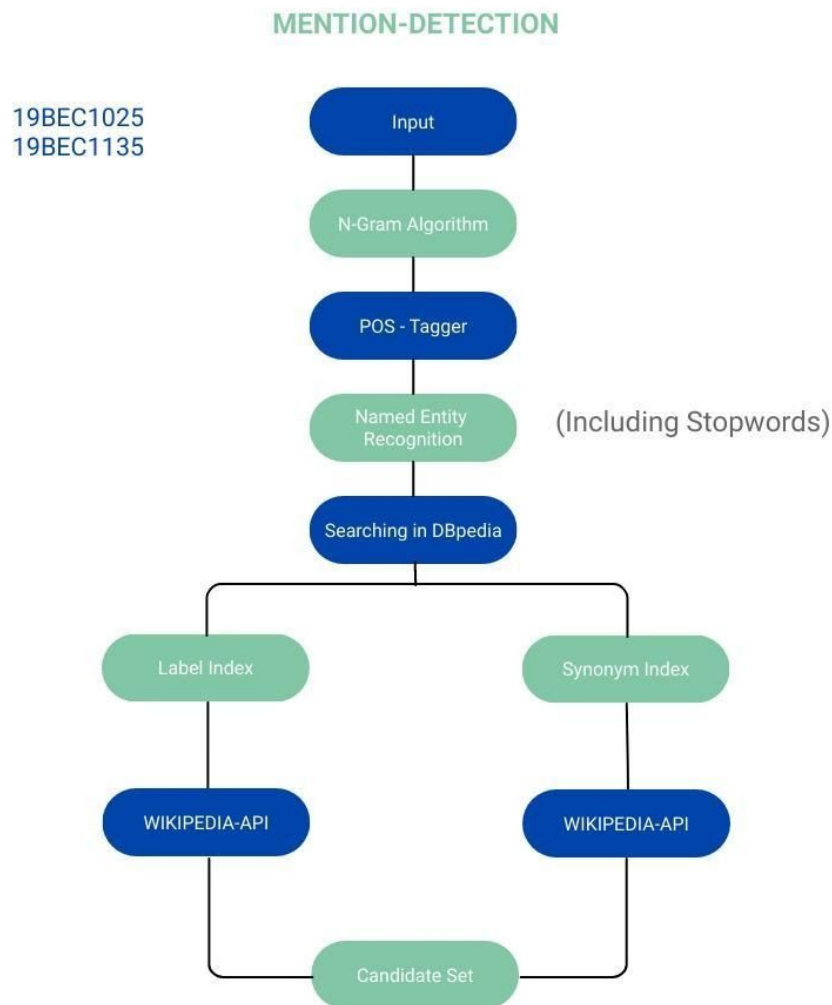
Example: if we search for Donald then we may get possible redirects like “Donald Duck” or “Donald Trump”



After searching in DBpedia we are going to make a change in the process. We are going to use wikipedia API for creating a list of top 5 Entities that are similar to mention using Popularity.

Which means we will be getting the most searched entities in candidate set. This will definitely increase the accuracy as the required mention most probably will be a well-known entity.

After adding this wikipedia API the flow of the Algorithm will be like this:



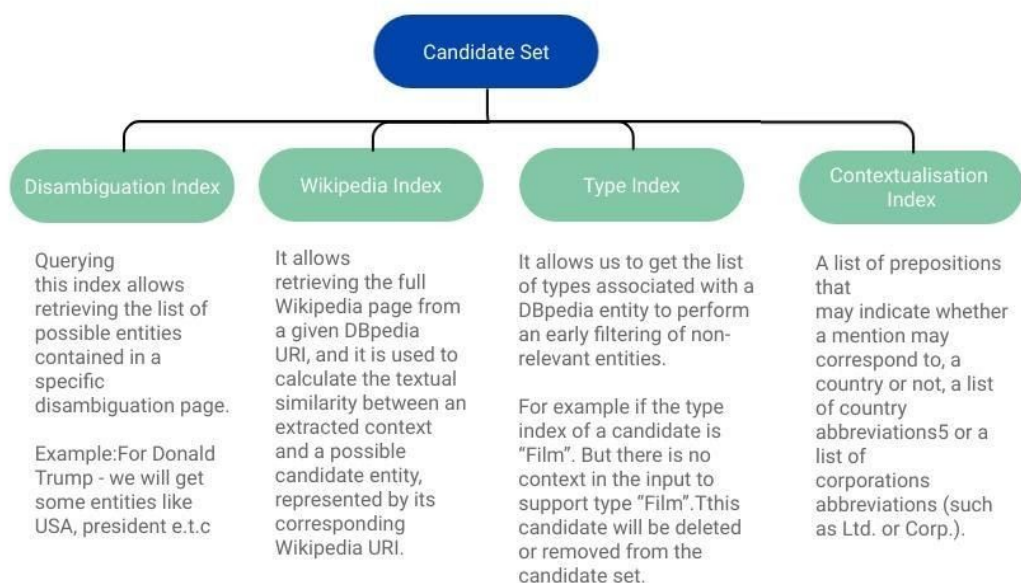
Example : For above example, if we search for “Donald” then we will be getting in order “Donald Trump” then “Donald Duck”. Because people searched for Donald Trump more than Donald Duck.

We will be also adding the disambiguation pages of each and every candidate in the candidate set. The limit of depth in adding the disambiguation pages is about 2 (As the number increase exponentially).

Now the candidate set will be having lots of candidates which are not relevant to given mention. So we will be using 4 different indexes which will help us to remove unwanted candidates.

## CANDIDATE-GENERATION

19BEC1025  
19BEC1135



1. Disambiguation index. Querying this index allows retrieving the list of possible entities contained in a specific disambiguation page. (which means it will give us the entities available in the disambiguation page.)

Example: For Donald Trump - we will get some entities like USA, American, president e.t.c

2. Wikipedia index. It allows retrieving the full Wikipedia page from a given DBpedia URI, and it is used to calculate the textual similarity between an extracted context and a possible candidate entity, represented by its corresponding Wikipedia URI.
3. Type index. This index allows getting the list of types associated with a DBpedia entity to perform an early filtering of irrelevant entities. We will be using this index to remove unwanted entities.

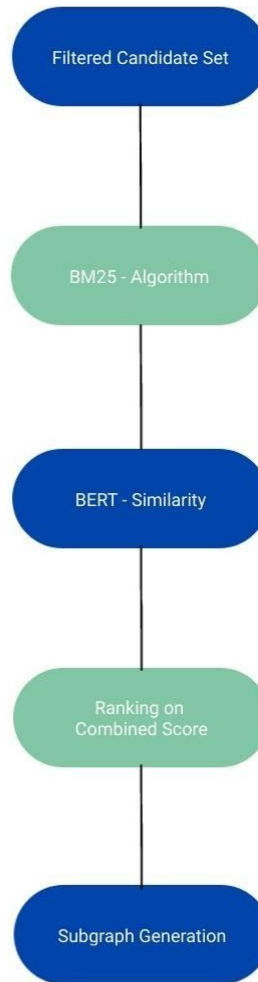
Example: if the type index of a candidate is about category "Film". But there is no context in the input that supports type "Film". Then this candidate will be deleted or removed from the candidate set.

4. Contextualization index. This index will help us to know whether the candidate is an organisation(Ltd. pvt.) or country or corporation e.t.c

After removing unwanted candidates, we will be using BM25 Ranking + BERT Similarity for improving the accuracy.



19BEC1025  
19BEC1135



Retrieval Phase:

$$BM25(Q, D) = \sum_{i \in Q} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1)f_i}{K + f_i} \cdot \frac{(k_2 + 1)q_{fi}}{k_2 + q_{fi}}$$

N — Size of the Collection of documents

$n_i$  — Number of documents in the collection containing query term  $t_i$

R — Relevant set size (i.e., number of documents judged relevant in collection)

$r_i$  — Number of judged relevant documents containing  $t_i$

$f_i$  — The frequency of term  $t_i$  in the document

$q_{fi}$  — The frequency of term  $t_i$  in the query

k1 — Determines how the tf(term frequency)component of the term weight changes as fi increases.

k2 — Determines how the tf (term frequency)component of the term weight changes as qfi increases

$K = k1((1-b) + b \cdot dl/avgdl)$ , where b is a parameter, dl is the length of the document, and avgdl is the average length of a document in the collection

BM25 depends on frequency of the term in the document .Which means if a term “t” is the most frequent word in a document “d” then d is the most relevant document for t

So (t,d) pair will be having high BM25 score.

Re-Retrieval Phase:

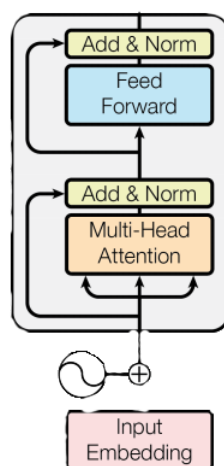
BERT-SIMILARITY Score:

BERT (Bidirectional Encoder Representations from Transformer.)

BERT is made up of series connection of ENCODERS.

ENCODERS are a type of neural networks.Which encodes the given data into smaller dimensions.

Encoder will eventually gets the core features of the input.For Similarity process we need our BERT model to understand the language,rather than understanding the words.



BERT uses Bidirectional Understanding which will be useful for our NLP tasks.

19BEC1025  
19BEC1135

## BI - DIRECTIONAL - UNDERSTANDING:

Example :

This is what normal NLP models do -

The movie is very \_\_\_\_\_ and terrible.

So it may predict the blank as "Good"

This is what BERT model does -

The movie is very \_\_\_\_\_ and terrible.

So after seeing word "terrible" it will predict "bad"

The BERT will understand the input and will try to predict the word bidirectionally. Which means if there is a sentence like "The movie is very \_\_\_\_\_ and terrible."

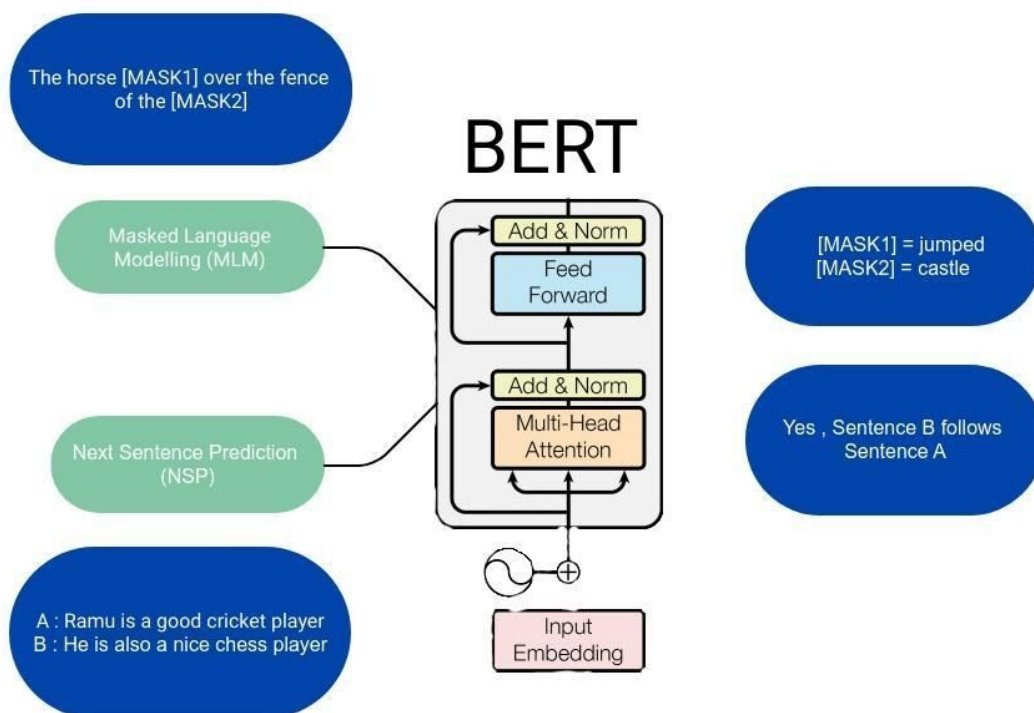
So, if i we give this sentence as input to normal NLP models they may predict the word as "good" because they will only consider the one way for prediction.

But,BERT goes bidirectional so when it comes across the word terrible it will predict the blank as "bad".

BERT is having a main advantage, that is BERT doesn't need any labeled data. It can even train on unlabeled data like wikipedia pages.

19BEC1025  
19BEC1135

### Bidirectional Encoder Representations from Transformers



Pre-training of BERT:

We will be training the BERT in two methods parallelly.

Method - 1 : Masked Language Modelling.

AIM: To know the context of the sentence -

We will be giving the BERT model, sentences with MASKS init. Like

“The horse [MASK1] over the fence of the [MASK2]”.

Then BERT model should predict the words under the MASKS. This type of approach is useful because it doesn't need labeled data.

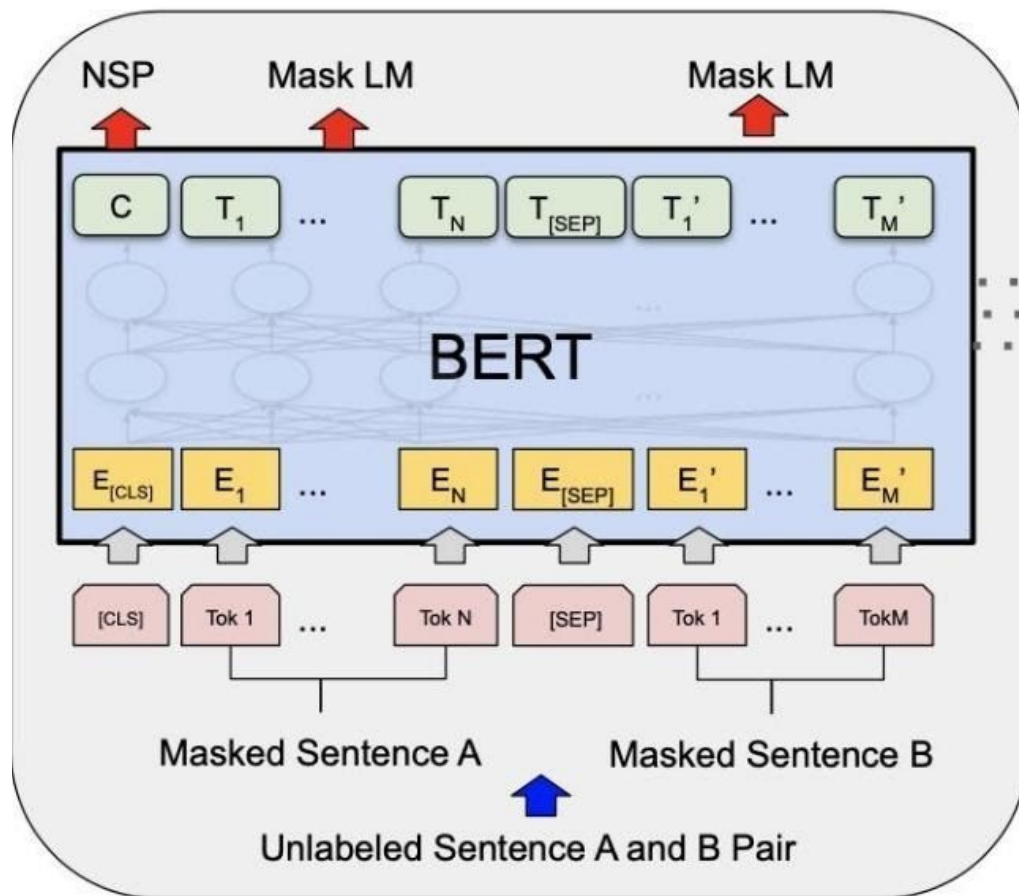
Method - 2 : Next Sentence Prediction:

AIM: To know the connection between sentences -

We will be giving two sentences like

- A. Ramu is a good cricket player.
- B. He is also a nice chess player.

Then it should say that sentence B follows sentence A. Which means in sentence B the pronoun HE refers to ramu.

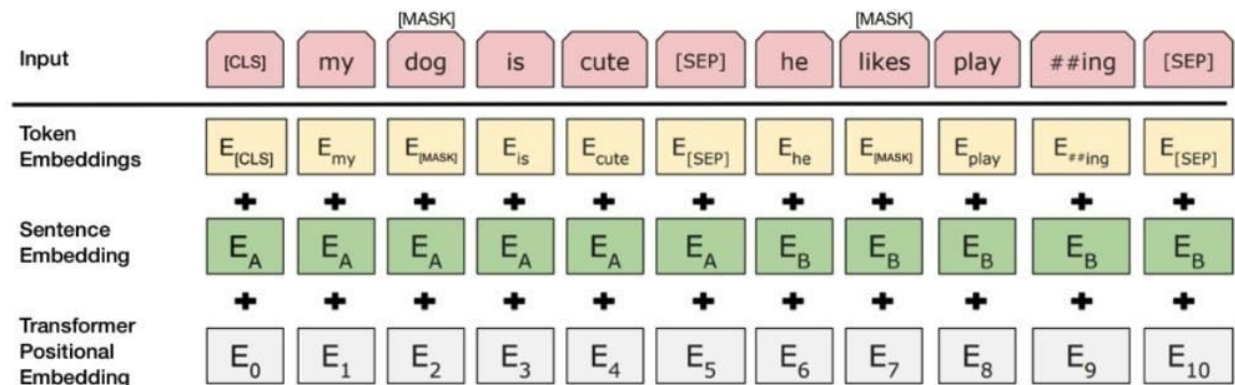


This is just a peek inside the working of BERT. The tokens which are

given to the BERT are MASKED. We will be converting them into embeddings. Then the whole process is same as we discussed in the previous page.

So now we are going to discuss about how we took the word embeddings for given word tokens.

We will be using the efficient Positional Embeddings:



Now this is how we are going to take the embeddings. We will be taking three major embeddings and adding them. Token embeddings are simple one-hot embedding. Then next is sentence Embedding that includes which sentence the word belongs to.

Positional embedding is the position of the word.

After training the BERT we can just give a mention and a wikipedia page of candidate in candidate set for producing the similarity score.

After finding both BM25 Ranking and BERT Similarity Score we will be calculating the Combined Score.

**Combined Score:**

**19BEC1025**

**19BEC1135**

**General Formula:**

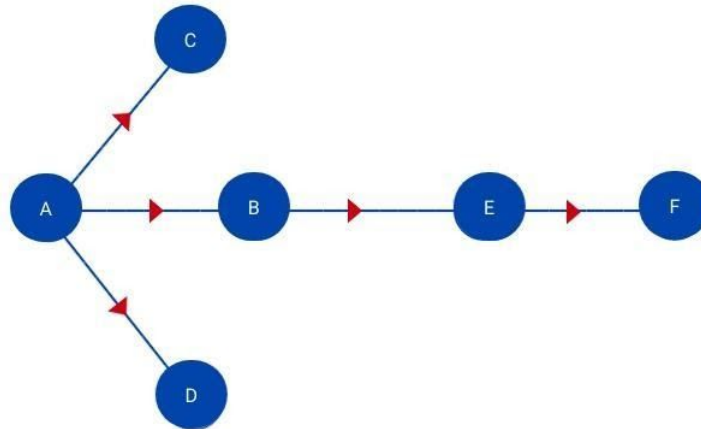
$$\text{Combined Score} = \text{BM25 Score} + \text{BERT Score}$$

*Based on this Combined Ranking we will be deciding the top 7 entities for subgraph generation.*

Based on the combined score we will be deciding the top 7 entities for sub graph generation.

But, forming a sub-graph with given top 7 entities is another challenge. We created our own centrality score for each and every node. That will help us to get over this challenge.

### Degree of Centrality:



The main problem here is A should be connected with F.  
So we should select one node from the three nodes C,B,D to traverse.  
For this we created Centrality Score, which helps in selecting the most popular node.

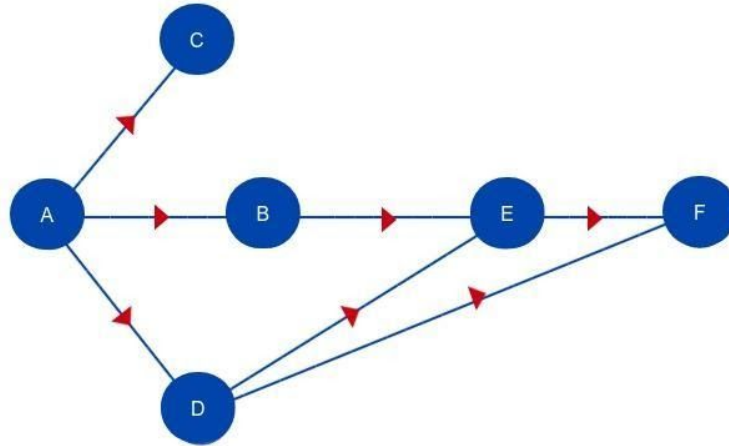
The main problem here is if we start traversing from starting point and also from ending point.  
We have lots of nodes to traverse. But we need to find the dense and popular node for traversing.

This is our Centrality Score Formula:



19BEC1025  
19BEC1135

### Degree of Centrality:



### Degree of Centrality:

19BEC1025  
19BEC1135

### General Formula:

$$\text{Centrality Score} = \frac{\phi}{n} + (1 - n) \theta$$

Here :

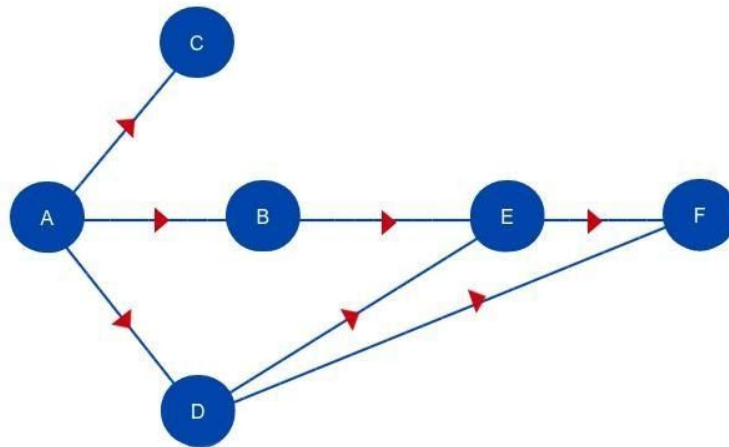
$\phi$  – Outgoing Connections  
 $\theta$  – Incoming Connections  
 $n$  – Normalized number of total nodes

*As the number of the nodes increases the importance to incoming nodes  
will be decreasing which is the thing we need*

We are testing this formula using a simple graph that has only 6 nodes connected.

19BEC1025  
19BEC1135

### Degree of Centrality:



For the above example:

19BEC1025  
19BEC1135

Centrality Score - Formula:

$$\begin{aligned} \text{Centrality Score} - A &= \frac{3}{0.6} + (1 - 0.6) 0 = 5 \\ \text{Centrality Score} - B &= \frac{1}{0.6} + (1 - 0.6) 1 = 2.0666 \\ \text{Centrality Score} - C &= \frac{0}{0.6} + (1 - 0.6) 1 = 0.4 \\ \text{Centrality Score} - D &= \frac{2}{0.6} + (1 - 0.6) 1 = 3.73 \\ \text{Centrality Score} - E &= \frac{1}{0.6} + (1 - 0.6) 2 = 2.4666 \\ \text{Centrality Score} - F &= \frac{0}{0.6} + (1 - 0.6) 2 = 0.8 \end{aligned}$$

Here :

$\phi$  - Outgoing Connections

$\theta$  - Incoming Connections

$n$  - normalized number of total nodes (6/10)

For normalizing the number of nodes we will be dividing it with 10 powers.

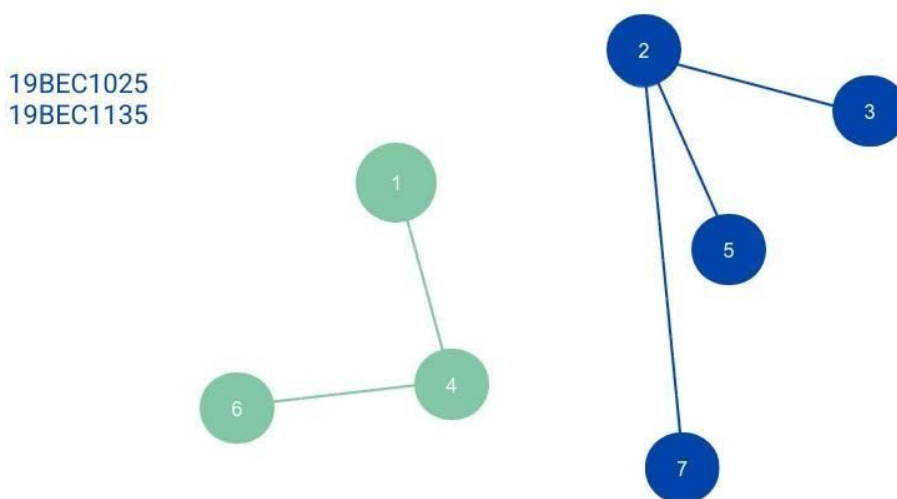
From the graph the importance in descending order should be  $A > D > E > B > F > C$

This is proved by the formula. In the formula we gave importance to the outgoing connections. Because if one node is connected to 3 other nodes with out going connections it will be good for us to traverse that node as we will be getting access to other 3 nodes.

For normalising the n-total number of nodes. We will be dividing it with nearest power of 10 (that is greater than total no of nodes).

This will make out centrality score to not be infinity at any instant.

After creating the sub-graphs ,if there is only one sub-graph it is good .But, if we get two or more sub-graphs we will be using our own critical score for deciding the best sub-graph.



For Selecting the best sub-graph between these two we will be using the below Score.

**Selecting Best Sub-graph:**

**19BEC1025**

**19BEC1135**

**General Formula:**

$$\text{Critical Score} = M + S$$

*Here :*

*M – Maximum of Centrality Score of all nodes.*

*S – Sum of Combined Scores of candidate nodes*

*if both of the subgraphs have the same score then we will be selecting the graph with max number of nodes.*

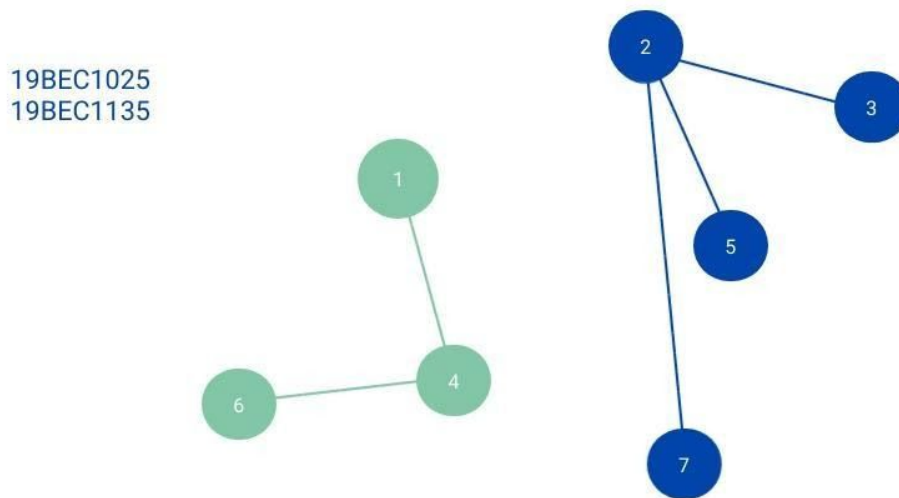
This Critical Score is addition of Maximum of Centrality Score of all nodes and Sum of Combined Scores of candidate nodes.

We are taking only these two because:

M - Maximum of centrality Score will be taken to find the most dense graph between these two sub-graphs.

S - From this we can find the sub-graph which is more relevant to the given mention.

For example:



For Sub-Graph 1:

$M1 = \text{Max}(\text{centrality score (all nodes of graph 1)})$

$S1 = \text{Sum}(\text{combined score}(1) + \text{combined score}(4) + \text{combined score}(6))$

Critical Score 1:  $C1 = M1 + S1$

This gives the critical-score-1 for the subgraph-1. Similarly we will be finding the critical-score-2 for subgraph-2.

After comparing the Two critical Scores we will be ending up with our Final output the Subgraph for given mention.

## **J component contribution (highlights):**

We made so many changes to the ABACO process in this project using BERT.

These are the changes and contributions from our side to the project.

### **1. Accuracy:**

ABACO:

ABACO used only BM25 Ranking for filtering the top 7 candidates.

OUR Project:

We used BM25 Ranking and also BERT Similarity Ranking for filtering the top 7 candidates.

### **2. Candidate Generation:**

ABACO:

ABACO used Label Index and Synonym Index for Candidate set

OUR Project:

We used Label Index, Synonym Index along with this we also used Wikipedia API which will provide the most popular entities which are relevant to the given mention. This will most probably give the nearest entity to the mention.

### **3. Sub-Graph Generation:**

ABACO:

ABACO used their own Degree of centrality for creating the sub-graph. But they found the score using all possible paths from the node.

OUR Project:

We used our own Centrality Score which targets on outgoing connections and incoming connections with respect to the number of nodes.

#### 4. Selecting the Sub-graph:

ABACO:

ABACO simply took the sub-graph with more number of nodes.

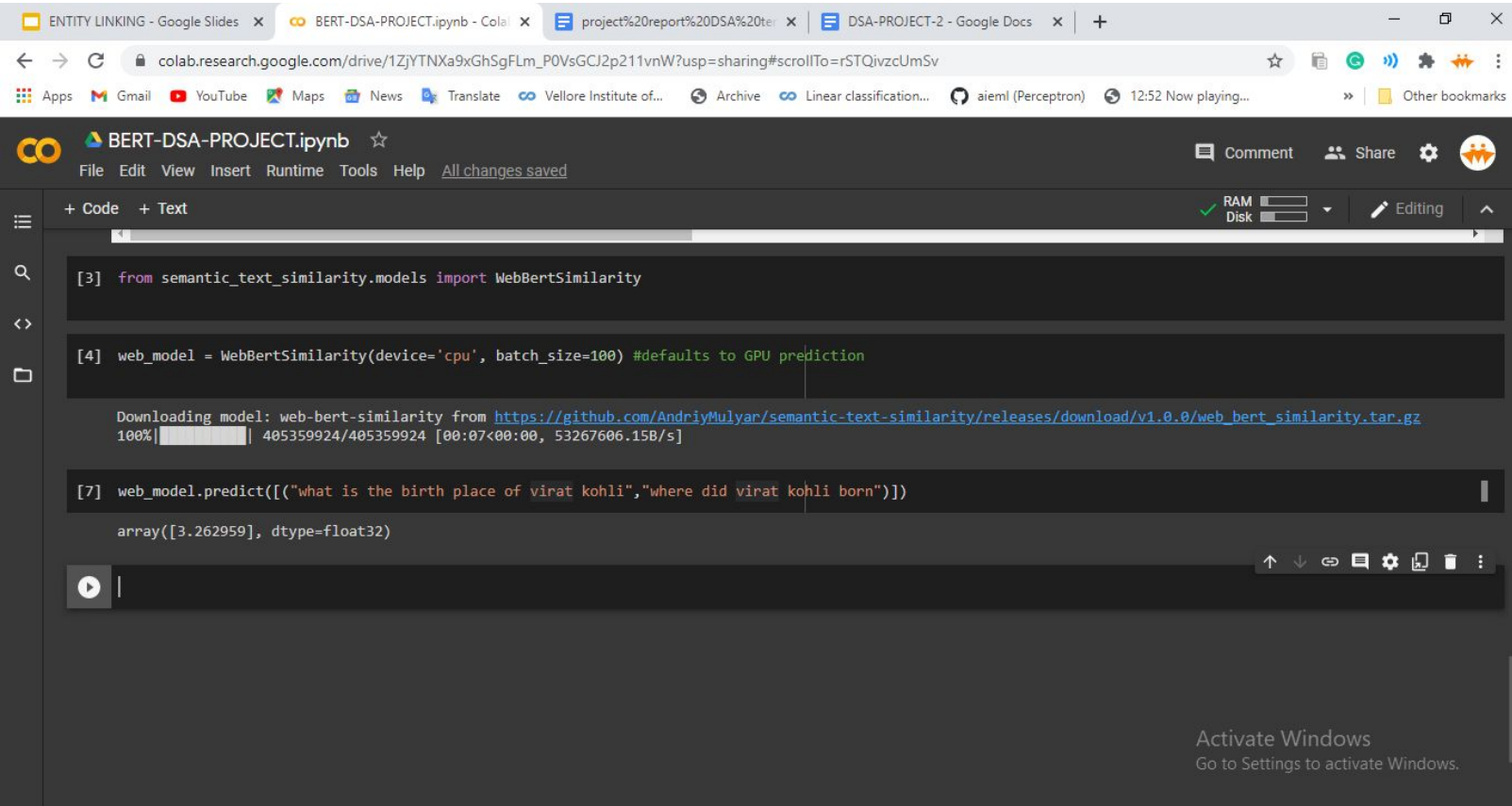
OUR Project:

We took our own critical score for each subgraph. We mainly focussed on Which subgraph is more Dense and popular. We also focussed on Which subgraph is more relevant to the given mention.

## Annexure:

This is how the BERT similarity works:

In this Demo we took the max similarity score as “5”. The BERT that we took here is of having upto 512 token input. But we can train a BERT beyond that limit too.



The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'ENTITY LINKING - Google Slides', 'BERT-DSA-PROJECT.ipynb - Colab', 'project%20report%20DSA%20te...', and 'DSA-PROJECT-2 - Google Docs'. The notebook's address bar shows the URL: [colab.research.google.com/drive/1ZjYTNXa9xGhSgFLm\\_POVsGCJ2p211vnW?usp=sharing#scrollTo=rSTQivzcUmSv](https://colab.research.google.com/drive/1ZjYTNXa9xGhSgFLm_POVsGCJ2p211vnW?usp=sharing#scrollTo=rSTQivzcUmSv). The notebook title is 'BERT-DSA-PROJECT.ipynb'. The code editor shows the following code:

```
[3] from semantic_text_similarity.models import WebBertSimilarity

[4] web_model = WebBertSimilarity(device='cpu', batch_size=100) #defaults to GPU prediction

Downloading model: web-bert-similarity from https://github.com/AndriyMulyar/semantic-text-similarity/releases/download/v1.0.0/web_bert_similarity.tar.gz
100%|██████████| 405359924/405359924 [00:07<00:00, 53267606.15B/s]

[7] web_model.predict([("what is the birth place of virat kohli","where did virat kohli born")])

array([3.262959], dtype=float32)
```

The output of the code is an array containing the value 3.262959. The interface also shows a RAM/Disk usage indicator and an 'Activate Windows' watermark at the bottom right.

For this Demo, we took two sentences like “what is the birth place of virat kohli?” and “where did virat kohli born?”

The BERT model predicted the semantic similarity between these two sentences is around “3.26” which is huge when the maximum is practically “4.89”.

The BERT can be trained with large datasets and also with large inputs. We can increase the accuracy of BERT by training it and also by reducing the change in weights of the neurons (called perceptrons).

This is our project based on Entity Linking using BERT similarity.



