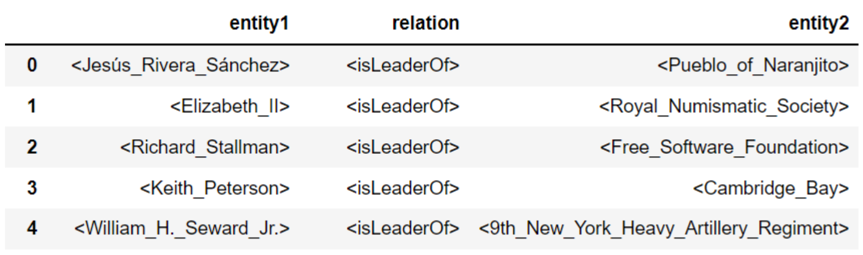
End-term Report for Reading Elective(4 Credits)

K.Sai Krishna
IMT2019045

Pre-midterm, I read about NER (Named Entity Recognition), NED (Named Entity Disambiguation), Coreference Resolution and Bert Embeddings. After that I used the Allen NLP tool for NER detection. The output generated by this tool uses some tagging scheme (like IOB, IOB2, BILOU etc…) . So, I read about different tagging schemes present and wrote a code snippet so that we will have tuples with the whole entity and its tag (instead of having prefix-tag for each word). So, by this point I had a setup that takes file or text input and gives the named entities as the output. What I did Post midterm is written below.

So, now our main aim is to build a pipeline for anonymization of a particular tag (say ‘LOC’). The anonymization task has to be done such that the third person shouldn’t recognize that the data is anonymized. For this to happen we have to replace the entities in our tag of interest with other entities such that the data still makes sense (on similar lines as before). So, one idea for possible replacement policy is to train Word2Vec model on **yago_dataset** and use the **most_similar**(word, n) function to find top-n most similar words of the given word.



	entity1	relation	entity2
0	<Jesús_Rivera_Sánchez>	<isLeaderOf>	<Pueblo_of_Naranjito>
1	<Elizabeth_II>	<isLeaderOf>	<Royal_Numismatic_Society>
2	<Richard_Stallman>	<isLeaderOf>	<Free_Software_Foundation>
3	<Keith_Peterson>	<isLeaderOf>	<Cambridge_Bay>
4	<William_H._Seward_Jr.>	<isLeaderOf>	<9th_New_York_Heavy_Artillery_Regiment>

Fig-1 shows how yago_dataset looks like.

The actual yago_dataset has 1,32,35,111 rows and when we perform inner join before passing them to Word2Vec model we will get 1,32,32,20,606 rows which the local system can’t handle. So I used the dataset with 3,00,000 rows only.

One observation one can make from Fig-1 is that the words in the entities are separated by underscores instead of space. This property of the dataset comes handy for us in future steps.

After finding the most_similar words of all the words (precisely entities) present in the yago_dataset, we can use these similar words for replacing the original word. But here while replacing we have to keep track of which word we are using to replace with another word as we should be able to revert back to the original state. So, we maintain a look-up table. We also use this look-up table to avoid clashes i.e., we take care of a particular word not being used as a replacement word for more than one word as this will create clashes while reverting back. So, now there has been a lot of describing the process which might not be as clear. So, to get a good clarity let's look at the pipeline.

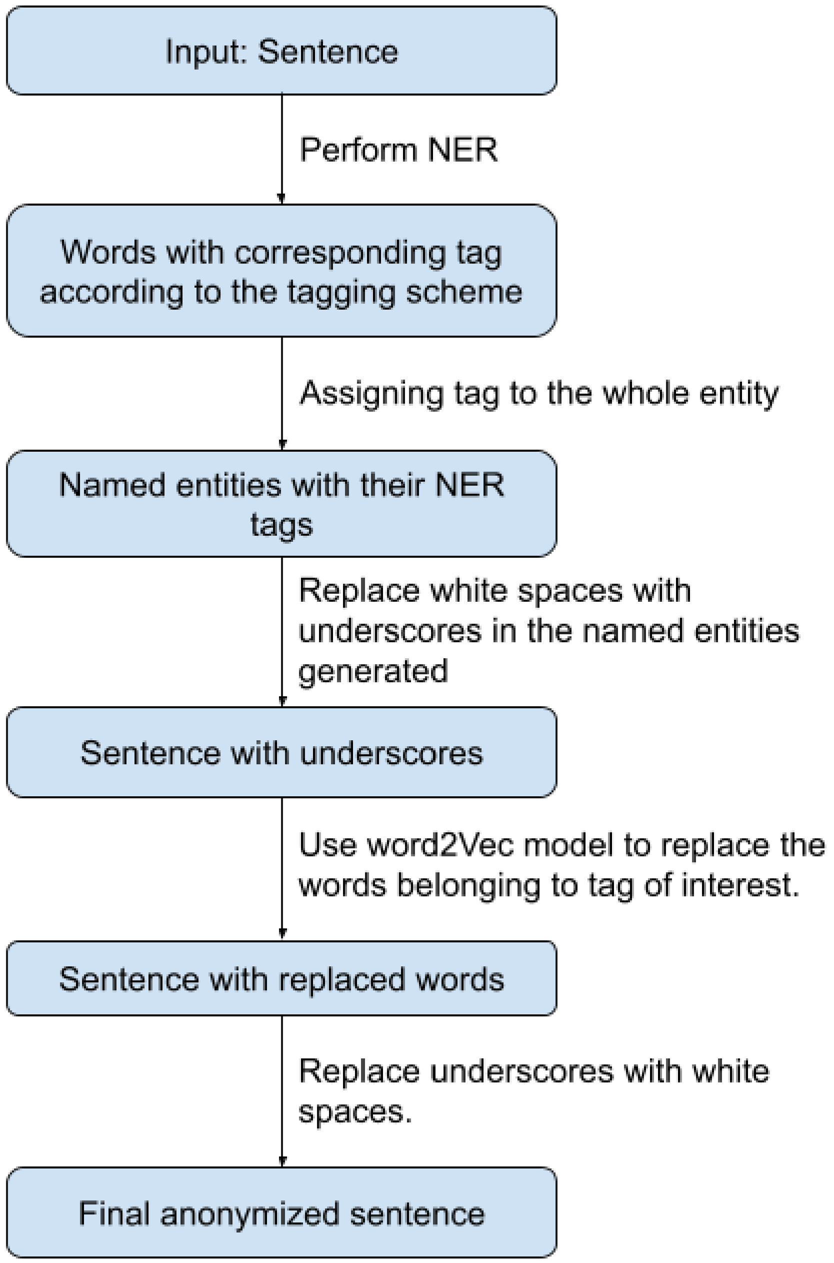

Input: Sentence
Perform NER
Words with corresponding tag according to the tagging scheme
Assigning tag to the whole entity
Named entities with their NER tags
Replace white spaces with underscores in the named entities generated
Sentence with underscores
Use word2Vec model to replace the words belonging to tag of interest.
Sentence with replaced words
Replace underscores with white spaces.
Final anonymized sentence

Fig-2 shows the pipeline of the data-anonymization problem.

Now let's look at the pipeline step-by-step with an example. In the example below our aim is to anonymize the ‘PER’ tag.

1. First we will have an input sentence to be anonymized.
   "**Albert Einstein** was born in Germany and lived in England.”

2. Now, let's apply NER on the input sentence given above using Allen NLP NER.
   [**('Albert', 'B-PER'), ('Einstein', 'L-PER')**, ('was', 'O'), ('born', 'O'), ('in', 'O'), ('Germany', 'U-LOC'), ('and', 'O'), ('lived', 'O'), ('in', 'O'), ('England', 'U-LOC'), ('.', 'O')]

3. As you can see the tag is assigned to a word (not an entity) using a tagging scheme which is BILOU in this case. To get tags without prefixes, we use the logic I wrote before midterm. After this step, the output will be as follows;
   [**('Albert Einstein', 'PER')**, ('was born in', 'O'), ('Germany', 'LOC'), ('and lived in', 'O'), ('England', 'LOC'), ('.', 'O')]

4. Now, to search for the most similar entities of our tag-of-interest (which is ‘PER’ in this example) Named entities, we first need to replace the white space in our tag-of-interest named entities with underscores as the named entities in the dataset contain underscores. After this step, the output will be as follows;
   [**'Albert_Einstein'**, 'was born in', 'Germany', 'and lived in', 'England', '.']

5. Now we will map the tag-of-interest named entities to other named entities using the most similar function and the look-up table. After this step, the output will be as follows;
   Friedrich_Kiel was born in Germany and lived in England .
   Here Albert_Einstein is replaced with Friedrich_Kiel.

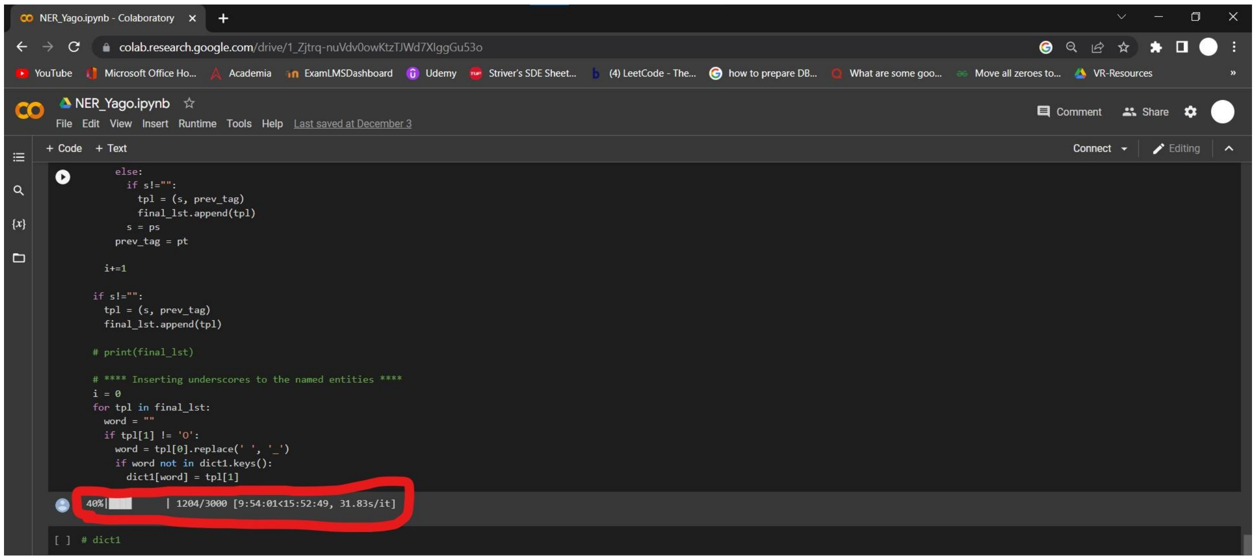
6. Finally, we will replace underscores with white spaces. After this step, the output will be as follows;
   Friedrich Kiel was born in Germany and lived in England .

This is how we set up a pipeline for this task. The replacement model used here (let’s name this model as **Replacement Model-1**) doesn’t perform very well as it is not guaranteed that the replacement word assigned has the same named entity tag as the original tag.

To overcome this problem, first we will perform NER on the yago_dataset. Now using this NER info of named entities in yago_dataset we append the yago_dataset with **‘is_a’** relationships.

> **Eg**: If NER for the entity ‘India’ is ‘LOC’, then the list [‘<India>’, ‘<is_a>’, ‘<LOC>’] is added to the original yago_dataset.

1. Ran NER on yago_dataset for 3,00,000 rows. It would take approximately *28 hours*, but google collab can't run for more than 10 hours continuously.

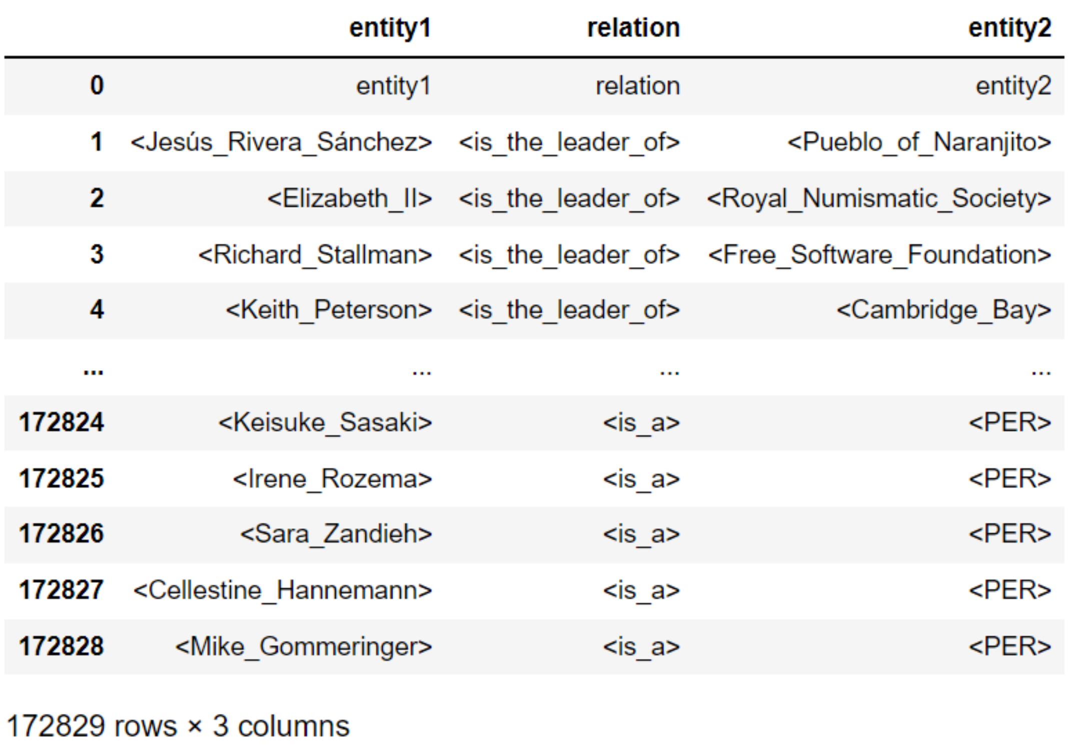


```
    else:
      if s!="":
        tpl = (s, prev_tag)
        final_lst.append(tpl)
      s = ps
    prev_tag = pt

  i+=1

if s!="":
  tpl = (s, prev_tag)
  final_lst.append(tpl)

# print(final_lst)

# **** Inserting underscores to the named entities ****
i = 0
for tpl in final_lst:
  word = ""
  if tpl[1] != 'O':
    word = tpl[0].replace(' ', '_')
    if word not in dict1.keys():
      dict1[word] = tpl[1]
```

```
40%|████      | 1204/3000 [9:54:01<15:52:49, 31.83s/it]
```

```
[ ] # dict1
```

2. So, decreased rows of yago_dataset from 3,00,000 to 80,000 and ran NER **to extract is_a relationships**. Now, it took *8 hrs* and generated the ner tags for all the entities present in that 80,000 rows. Now, the new dataframe is as shown in Fig-3;



	entity1	relation	entity2
0	entity1	relation	entity2
1	<Jesús_Rivera_Sánchez>	<is_the_leader_of>	<Pueblo_of_Naranjito>
2	<Elizabeth_II>	<is_the_leader_of>	<Royal_Numismatic_Society>
3	<Richard_Stallman>	<is_the_leader_of>	<Free_Software_Foundation>
4	<Keith_Peterson>	<is_the_leader_of>	<Cambridge_Bay>
...	...	...	...
172824	<Keisuke_Sasaki>	<is_a>	<PER>
172825	<Irene_Rozema>	<is_a>	<PER>
172826	<Sara_Zandieh>	<is_a>	<PER>
172827	<Cellestine_Hannemann>	<is_a>	<PER>
172828	<Mike_Gommeringer>	<is_a>	<PER>

172829 rows × 3 columns

Fig-3

Replacement Model-2: In this model we will now re-train the Word2Vec model using the newly generated dataset and then repeat the same pipeline.

The only change for this model and the previous model is the dataset used for Word2Vec model. The results for both the models for 3 sample entities are shown in the below table. Note that as the Replacement Model-2 ran using 80,000 rows so I ran the Replacement Model-1 again using 80,000 rows for the sake of comparison.

Word from vocab	**Replacement Model-1**	**Replacement Model-2**
'India' (LOC)	owns isCitizenOf de/Petras_Čimbaras MTV_Southeast_Asia Virgilijus_Kačinskas Carolina_Gaitán Česlovas_Kundrotas Mohieddin_Fikini Jhon_Lucumí Kim_Poor	Mumtaz_Ahmed_Khannt-1_(humanitarian) Amit_Khanna_(photographer S._K._Roongta P._K._Sreemathy Theda_Nelson_Clarke Gouri_Sankar_Dutta 'fr/Onet_(entreprise) Manibhai_Ramjibhai_Chaudhary de/Pablo_Mariaselvam Siddappa_Kambli
'South_Korea' (LOC)	Gyeongju_Tower Jeju_Baseball_Stadium Korea_Aerospace_Research_Institut e Jeonnam_Stadium Gangjin_Baseball_Park Qatar Malyshev_Factory Kiev_Arsenal	Serbia XHHCU-TDT La_Linda_International_Bridge Malyshev_Factory Jeju_Baseball_Stadium Jeonnam_Stadium Korea_Aerospace_Research_Institute 'Photoprylad Gangjin_Baseball_Park
'Sony' (ORG)	West_Japan_Railway_Company East_Japan_Railway_Company CBS_Corporation The_Master_Trust_Bank_of_Japan Japan_Trustee_Services_Bank Charter_Communications Central_Japan_Railway_Company National_Amusements Apple_Inc Time_Warner	West_Japan_Railway_Company East_Japan_Railway_Company Japan_Trustee_Services_Bank The_Master_Trust_Bank_of_Japan Nippon_Life Central_Japan_Railway_Company Sumitomo_Mitsui_Banking_Corporation State_Street_Corporation SSBT_OD05_Omnibus Mizuho_Bank

As you can see the outputs in the case of 'South_Korea' are improved but there is no improvement in the case of the other two models. This might be because we are running on a very small subset of dataset. I think the outputs will be improved if we do this on the whole dataset.

Future Scope:

Try and enhance the replacement model which presently is using Word2Vec similarity function.

Some ideas are;

1. Run it for the whole dataset.
2. Write a **custom similarity function** which calculates similarity scores **among the same tag**. The similarity function can take a dictionary as an input which has word as its key and in values it will have two fields which are;
    a. Word embedding of that entity.
    b. NER tag of that entity.
3. Search for the first occurrence of the word in the most similar words list which has the same ‘NER’ tag and replace it with that word (Naive approach).