دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

Information Retrieval Course

Fall 2023

IR-HW02

Koorosh Khavari Moghaddam

koorosh.k@aut.ac.ir

koorosh.khavari.m@gmail.com

Student ID: 401131023

Contents

Preprocessing Data

Preprocessing Document Dataset

```
1  # preprocessing : tokenization, lower case, stop-word removal
2  term_doc = {}
3  doc_term = {}
4  for index, row in docs.iterrows():
5      term_doc[row['doc_id']] = []
6      tokens_list = re.split(r'\s+', re.sub(r'[^\w\s]',' ',row['document'].lower()))
7      for token in tokens_list:
8          if token in doc_term:
9              doc_term[token] += [row['doc_id']]
10         elif (((len(token)>=2) and any(not char.isdigit() for char in token)) or (len(token)>=4)) and (token not in stops):
11             doc_term[token] = [row['doc_id']]
12         else: continue
13         term_doc[row['doc_id']] += [token]
```

[6,10] All punctuations and irrelevant numbers sequences (length lower than 4) are removed from the text and the text is converted to lowercase, tokenization is performed by [6] splitting the text based on white spaces. Also, [6] all the stop words are removed from the corpus, the list of the stop words is imported as a raw text file.

TF Information

TF vectors of the words in the documents and queries are calculated similar to the problem set 1, we only kept Term Frequencies [22,47] as an estimate of probability of word given document distribution. The dimensionality of the vectors is 10k to achieve perfect result. This choice may represent other approaches superior to the BERT model in the evaluation part, but keep in mind that those results are happened given we have perfect information about the data and are not reliable.

```
1  # find most frequent terms
2  # constants
3  DIC_LENGTH = 10000
4  NUM_RELATD = 10
5  EPSILON = 1e-10
6
7  doc_term = dict(sorted(doc_term.items(), key=lambda item: len(item[1])))
8  # make dictionary of 1000 important words
9  dictionary = {term:index for index,term in enumerate(list(doc_term.keys())[0:DIC_LENGTH])}
10 # make dictionary-term tf-idf matrix for docs
11 # this operation will result the DT matrix of shape(1000,750), which is 750 doc vectors in dictionary space
12 TD = np.zeros((len(dictionary),len(term_doc)))
13 for index,(doc,terms) in enumerate(term_doc.items()):
14     doc_in_dict = {}
15     for term in terms:
16         if term in dictionary:
17             if term in doc_in_dict:
18                 doc_in_dict[term] += 1
19             else:
20                 doc_in_dict[term] = 1
21     for term in doc_in_dict.keys():
22         TD[dictionary[term],index] = (doc_in_dict[term]/len(doc_in_dict))
23 # make dictionary-term tf-idf matrix for queries
24 # this operation will result the DT matrix of shape(1000,50), which is 50 query vectors in dictionary space
25 query_term = {}
26 term_query = {}
27 for index, row in queries.iterrows():
28     term_query[row['query_id']] = []
29     tokens_list = re.split(r'\s+', re.sub(r'[^\w\s]',' ',row['query'].lower()))
30     for token in tokens_list:
31         if token in dictionary:
32             if token in query_term:
33                 query_term[token] += [row['query_id']]
34             else:
35                 query_term[token] = [row['query_id']]
36             term_query[row['query_id']] += [token]
37
38 TQ = np.zeros((len(dictionary),queries.shape[0]))
39 for index,(query,terms) in enumerate(term_query.items()):
40     query_in_dict = {}
41     for term in terms:
42         if term in query_in_dict:
43             query_in_dict[term] += 1
44         else:
45             query_in_dict[term] = 1
46     for term in query_in_dict.keys():
47         TQ[dictionary[term],index] = (query_in_dict[term]/len(query_in_dict))
```

Information Retrieval using Unigram Model

```python
# Jelinek-Mercer smoothing : given λ, will calculate and return smoothed TD vector
def JMS(TD=TD, λ=0.2): return λ*(TD/(np.sum(TD,axis=0)+EPSILON)) + (1-λ)*(np.sum(TD,axis=1)/np.sum(TD)).reshape(-1,1)
# Unigrams assumes independence of query words given document model
def Unigram(TD=TD,TQ=TQ, λ=0.2): return np.argsort(-np.exp(np.einsum("ij,ik->jk",np.log(JMS(TD=TD, λ=λ)),TQ/np.sum(TQ,axis=0))),axis=0).T[:, :NUM_RELATD]
```

Whole algorithm can be described in two lines. As for the Jelinek-Mercer smoothing part which is described in [2], we simply added up weighted sum of the term frequencies inside the document and the term frequencies inside the whole collection. The unigram probabilities then can be calculated as shown in [4] by a simple trick of rescaling the data. The unigram model can be described as $P(Q|D) = \prod_{q\in Q} P(q|D)^{TF_{q,Q}}$ (refer to IR_Lec5, page 17). We estimated the $P(q|D) \sim \frac{TF_{q,D}}{|D|}$ and we also know the value for the $TF_{q,Q}$ in the term frequency tensors. Therefor simple by representing the $P(q|D)$ in the logarithm space and perform Einstein sum on this tensor and $TF_{q,Q}$ and then transform it back with exponential as in [4], we can exactly describe the $P(Q|D)$.

As for the optimal value for λ parameter, we can examine the results of our evaluation part:

```
[Unigram_λ=0.1   MRR]: 0.72      [Unigram_λ=0.1   MAP]: 0.48      [Unigram_λ=0.1   P@5]: 0.55      [Unigram_λ=0.1   P@10]: 0.48
[Unigram_λ=0.2   MRR]: 0.74      [Unigram_λ=0.2   MAP]: 0.49      [Unigram_λ=0.2   P@5]: 0.56      [Unigram_λ=0.2   P@10]: 0.49
[Unigram_λ=0.3   MRR]: 0.74      [Unigram_λ=0.3   MAP]: 0.48      [Unigram_λ=0.3   P@5]: 0.56      [Unigram_λ=0.3   P@10]: 0.48
[Unigram_λ=0.4   MRR]: 0.74      [Unigram_λ=0.4   MAP]: 0.46      [Unigram_λ=0.4   P@5]: 0.55      [Unigram_λ=0.4   P@10]: 0.46
[Unigram_λ=0.5   MRR]: 0.74      [Unigram_λ=0.5   MAP]: 0.45      [Unigram_λ=0.5   P@5]: 0.55      [Unigram_λ=0.5   P@10]: 0.45
[Unigram_λ=0.6   MRR]: 0.73      [Unigram_λ=0.6   MAP]: 0.45      [Unigram_λ=0.6   P@5]: 0.53      [Unigram_λ=0.6   P@10]: 0.45
[Unigram_λ=0.7   MRR]: 0.73      [Unigram_λ=0.7   MAP]: 0.45      [Unigram_λ=0.7   P@5]: 0.53      [Unigram_λ=0.7   P@10]: 0.45
[Unigram_λ=0.8   MRR]: 0.73      [Unigram_λ=0.8   MAP]: 0.45      [Unigram_λ=0.8   P@5]: 0.53      [Unigram_λ=0.8   P@10]: 0.45
[Unigram_λ=0.9   MRR]: 0.75      [Unigram_λ=0.9   MAP]: 0.45      [Unigram_λ=0.9   P@5]: 0.53      [Unigram_λ=0.9   P@10]: 0.45
```

As shown in the above figure, the joint vote of the evaluation metrics is $\lambda = 0.2$. Result of the ranking with the said parameter is shown below:

```
[Query]: what is the origin of COVID-19
[RELATED DOCS INDECES]: [637 690 668 389 425 557 435 748 609 278]
[RELATED DOCS IDS]: ['lj8t52yl', '2y452utz', '0cq5ee1i', '0xkz36bj', '22fc1qly', '0m5mc320', '41378qru', '105q161g', '1mjaycee', '1vkz0b0o']
TOP FIVE RESULT FOR 1'st QUERY
[lj8t52yl]: The origin of the SARS-CoV-2 virus remains enigmatic. It is likely to be a continuum resulting from inevitable mutations and recombination e
[2y452utz]: A novel coronavirus strain 2019-nCoV has caused a rapid global pandemic-COVID-19. Scientists have taken onto the task of characterizing this
[0cq5ee1i]: INTRODUCTION: SARS-CoV-2 was first detected in December 2019 in the Chinese city of Wuhan and has since spread across the world. At present,
[0xkz36bj]: INTRODUCTION: SARS-CoV-2 was first detected in December 2019 in the Chinese city of Wuhan and has since spread across the world. At present,
[22fc1qly]: Coronaviruses are the well-known cause of severe respiratory, enteric and systemic infections in a wide range of hosts including man, mammal
[QREL]:     query_id    doc_id
0           1  005b2j4b
1           1  0chuwvg6
2           1  0t2a5500
3           1  0y34yxlb
4           1  105q161g
5           1  11edrkav
6           1  1abp6oom
7           1  1mjaycee
8           1  1sq2uvur
9           1  22fc1qly
10          1  23y18so0
11          1  24yavi1w
12          1  25va2cvt
13          1  262bcl7h
14          1  214xxu3v
```

Information Retrieval using Bigram Model

```python
# Jelinek-Mercer smoothing : given λ_1, λ_2, will calculate and return smoothed TTD vector
def JMS(TTD=TTD, TD=TD, λ_1=0.2, λ_2=0.2): return λ_1*TTD + (λ_2*(TD/(np.sum(TD,axis=0)+EPSILON)) + (1-λ_1-λ_2)*(np.sum(TD,axis=1)/(np.sum(TD)+EPSILON)).reshape(-1,1)).reshape((DIC_LENGTH,1,-1))
# Bigrams assumes independence of query words given document model and one previous term if exists
def Bigram(TTD=TTD, TTQ=TTQ, TD=TD, TQ=TQ, λ_1=0.2, λ_2=0.2): return np.argsort(-np.einsum("ijd,ijq->dq",np.log(JMS(TTD=TTD, TD=TD, λ_1=λ_1, λ_2=λ_2)+1e-12),TTQ), axis=0).T[:, :NUM_RELATD]
```

The whole bigram algorithm is exactly like the unigram one with one simple caveat which is the estimation of $P(Q|D)$, for the simplicity of the implementation, I just used the ordered conditional probabilities $P(w_i|w_{i-1}, D)$ and dropped the first term unigram. In order to estimated the ordred conditional prbability term, we are going to sacrifice the precision slightly in order to achive code simplicity, instead of a dictionary with the size of 10k, I used a 700 entry dictionary, given this simplification, we can represent the ordered conditinal probability terms in documents and queries as a tensors of the shapes (700,700,750) and (700,700,50) respectively, the first two dimensions are the size of the dictionary and the third one is the total number of the documents and queries. Again given the simplified formula $P(Q|D) = \prod_{q\in Q} P(q_i|q_{i-1}, D)^{TF_{q_i,q_{i-1},Q}}$ we can estimate the conditional

[Bigram, λ_1=0.6, λ_2=0.1 MRR]: 0.65	[Bigram, λ_1=0.6, λ_2=0.1 MAP]: 0.39	[Bigram, λ_1=0.6, λ_2=0.1 P@5]: 0.44	[Bigram, λ_1=0.6, λ_2=0.1 P@10]: 0.39
[Bigram, λ_1=0.6, λ_2=0.2 MRR]: 0.69	[Bigram, λ_1=0.6, λ_2=0.2 MAP]: 0.40	[Bigram, λ_1=0.6, λ_2=0.2 P@5]: 0.48	[Bigram, λ_1=0.6, λ_2=0.2 P@10]: 0.40
[Bigram, λ_1=0.6, λ_2=0.3 MRR]: 0.71	[Bigram, λ_1=0.6, λ_2=0.3 MAP]: 0.40	[Bigram, λ_1=0.6, λ_2=0.3 P@5]: 0.51	[Bigram, λ_1=0.6, λ_2=0.3 P@10]: 0.40
[Bigram, λ_1=0.7, λ_2=0.1 MRR]: 0.66	[Bigram, λ_1=0.7, λ_2=0.1 MAP]: 0.39	[Bigram, λ_1=0.7, λ_2=0.1 P@5]: 0.45	[Bigram, λ_1=0.7, λ_2=0.1 P@10]: 0.39
[Bigram, λ_1=0.7, λ_2=0.2 MRR]: 0.70	[Bigram, λ_1=0.7, λ_2=0.2 MAP]: 0.40	[Bigram, λ_1=0.7, λ_2=0.2 P@5]: 0.49	[Bigram, λ_1=0.7, λ_2=0.2 P@10]: 0.40
[Bigram, λ_1=0.8, λ_2=0.1 MRR]: 0.66	[Bigram, λ_1=0.8, λ_2=0.1 MAP]: 0.39	[Bigram, λ_1=0.8, λ_2=0.1 P@5]: 0.45	[Bigram, λ_1=0.8, λ_2=0.1 P@10]: 0.39
[Bigram, λ_1=0.9, λ_2=0.1 MRR]: 0.66	[Bigram, λ_1=0.9, λ_2=0.1 MAP]: 0.38	[Bigram, λ_1=0.9, λ_2=0.1 P@5]: 0.44	[Bigram, λ_1=0.9, λ_2=0.1 P@10]: 0.38

probability as shown in [4]. As for the smoothing part, it's performed in [2] as instructed in the problem.

Like the previous part, by comparing different values for λ_1 and λ_2, all the metrics suggest that $\lambda_1, \lambda_2 = 0.6, 0.3$ yeild better performance compared to other condiddates.

Information Retreival using Word2Vec Model

```python
from gensim.models import Word2Vec
model = Word2Vec(list(term_doc.values())+list(term_query.values()), min_count=1, sg=1, vector_size=200, epochs=20)
# Arithmatic Mean vectors
doc_vec = np.array([np.array([model.wv[term] for term in terms]).mean(axis=0) for (doc,terms) in term_doc.items()])
query_vec = np.array([np.array([model.wv[term] for term in terms]).mean(axis=0) for (query,terms) in term_query.items()])

# Weighted Mean vectors
weighted_doc_vec = np.array([np.array([model.wv[term]*TD[dictionary[term],index] for term in terms]).mean(axis=0) for index,(doc,terms) in enumerate(term_doc.items())])
weighted_query_vec = np.array([np.array([model.wv[term]*TQ[dictionary[term],index] for term in terms]).mean(axis=0) for index,(query,terms) in enumerate(term_query.items())])

AW2V = np.argsort(-np.einsum("qi,di->qd",query_vec,doc_vec)/(np.linalg.norm(query_vec, axis=1).reshape((-1,1))*np.linalg.norm(doc_vec, axis=1).reshape((1,-1))), axis=1)[:, :NUM_RELATD]
WW2V = np.argsort(-np.einsum("qi,di->qd",weighted_query_vec,weighted_doc_vec)/(np.linalg.norm(weighted_query_vec, axis=1).reshape((-1,1))*np.linalg.norm(weighted_doc_vec, axis=1).reshape((1,-1))), axis=1)[:, :NUM_RELATD]

check_results(0, WW2V)
```

Using the genism library as suggested in the homework, Word2Vec model trained using 20 epochs and a dimension of 200 for representing the words inside the corpus consisting of documents and queries [2]. After that, as instructed in the problem, we calculate two representations of the documents and queries in the vector space, first one without any prior knowledge (or arithmetic case in the first section of the problem) [4,5] and the other with the prior knowledge TF-IDF for words (or the weighted case in the second section of the problem) [8,9]. As we will see in the result section, assuming a precise prior for the word2vec model will results in an incredibly accurate model which surpasses the BERT at least for the case of limited documents and queries. In order to calculate the cosine similarities in the vector space, just like the vector space in problem set 1, we calculate dot product of vectors normalized by their norm production [11,12]

Result of arithmetic Word2Vec is shown below:

```
[Query]: what is the origin of COVID-19
[RELATED DOCS INDECES]: [276 721 425 690 609 435 420 131 351 683]
[RELATED DOCS IDS]: ['2lxs9laj', '1emlkii0', '22fc1qly', '2y452utz', '1mjaycee', '41378qru', '12sbikmx', '03pd9jtn', '0uvzy48c', '4r0t3q7j']
TOP FIVE RESULT FOR 1'st QUERY
[2lxs9laj]: Coronavirus disease 2019 (COVID-19), which causes serious respiratory illness such as pneumonia and lung failure, was first reported
[1emlkii0]: COVID-19 is a bluff
[22fc1qly]: Coronaviruses are the well-known cause of severe respiratory, enteric and systemic infections in a wide range of hosts including man,
[2y452utz]: A novel coronavirus strain 2019-nCoV has caused a rapid global pandemic-COVID-19. Scientists have taken onto the task of characterizi
[1mjaycee]: The coronavirus disease 19 (COVID-19) is a highly transmittable and pathogenic viral infection caused by severe acute respiratory syr
[QREL]:      query_id    doc_id
0           1  005b2j4b
1           1  0chuwvg6
2           1  0t2a5500
3           1  0y34yxlb
4           1  105q161g
5           1  11edrkav
6           1  1abp6oom
7           1  1mjaycee
8           1  1sq2uvur
9           1  22fc1qly
10          1  23y18so0
11          1  24yavi1w
12          1  25va2cvt
13          1  262bcl7h
14          1  214xxu3v
```

Result of weighted Word2Vec is shown below:

```
[Query]: what is the origin of COVID-19
[RELATED DOCS INDECES]: [396  20 690 748 425 736 490 122 217 701]
[RELATED DOCS IDS]: ['24yavi1w', '0t2a5500', '2y452utz', '105q161g', '22fc1qly', '0chuwvg6', '1sq2uvur', '1fxrmuzl', 'vk8s1f23', 'ig0rnbqb']
TOP FIVE RESULT FOR 1'st QUERY
[24yavi1w]: Mutation and adaptation have driven the co-evolution of coronaviruses (CoVs) and their hosts, including human beings, for thousands of years. Be
[0t2a5500]: The Chinese rufous horseshoe bat (Rhinolophus sinicus) has been suggested to carry the direct ancestor of severe acute respiratory syndrome (SAR
[2y452utz]: A novel coronavirus strain 2019-nCoV has caused a rapid global pandemic-COVID-19. Scientists have taken onto the task of characterizing this new
[105q161g]: A number of virological, epidemiological and ethnographic arguments suggest that COVID-19 has a zoonotic origin. The pangolin, a species threate
[22fc1qly]: Coronaviruses are the well-known cause of severe respiratory, enteric and systemic infections in a wide range of hosts including man, mammals, f
[QREL]:      query_id    doc_id
0           1  005b2j4b
1           1  0chuwvg6
2           1  0t2a5500
3           1  0y34yxlb
4           1  105q161g
5           1  11edrkav
6           1  1abp6oom
7           1  1mjaycee
8           1  1sq2uvur
9           1  22fc1qly
10          1  23y18so0
11          1  24yavi1w
12          1  25va2cvt
13          1  262bcl7h
14          1  214xxu3v
```

Information Retreival using BERT model

```
1  tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
2  model = BertModel.from_pretrained('bert-base-uncased')
3  doc_ids = [torch.tensor([tokenizer.encode(sentence, add_special_tokens=True)])[:, :512] for sentence in list(docs['document'])]
4  query_ids = [torch.tensor([tokenizer.encode(sentence, add_special_tokens=True)])[:, :512] for sentence in list(queries['query'])]
5  with torch.no_grad():
6      # results = []
7      for doc_id in tqdm(doc_ids):
8          try: results += [model(doc_id).last_hidden_state.mean(dim=1).squeeze()]
9          except: print(f"[error] : {doc_id}")
10     doc_vec = torch.stack(results)
11     results = []
12     for query_id in tqdm(query_ids):
13         try: results += [model(query_id).last_hidden_state.mean(dim=1).squeeze()]
14         except: print(f"[error] : {query_id}")
15     query_vec = torch.stack(results)
16 import pickle
17 with open ("bert_vectors.pth", "wb") as f:
18     pickle.dump((doc_vec, query_vec), f)
```

Small pretrained BERT model is used to represent the documents and queries in vector space of shape (768,) [1-15]. The result vectors are stored in a pickle file for the convenience [17-18], just load and use them in case of verifying the result.

```
DIC_LENGTH = 10000
NUM_RELATD = 10
EPSILON = 1e-10
BERTR = torch.argsort(-torch.einsum("qi,di->qd",query_vec,doc_vec)/(torch.norm(query_vec, dim=1).reshape((-1,1))*torch.norm(doc_vec, dim=1).reshape((1,-1))), axis=1)[:, :NUM_RELATD].numpy()
check_results(1, BERTR)
```

We calculate the cosine similarities like all other previous parts, by normalizing the dot product of vector representation with respect to their norm product.

BERT ranking results are shown below:

```
[Query]: how does the coronavirus respond to changes in the weather
[RELATED DOCS INDECES]: [602 104 261 637 717 285 701 559 139 476]
[RELATED DOCS IDS]: ['03s9spbi', '431ksdno', '15slu3kk', 'lj8t52yl', '0pujch9v', '04awj06g', 'ig0rnbqb', '6vln3erl', '0vlzwksu', '8r6u3e3i']
TOP FIVE RESULT FOR 1'st QUERY
[03s9spbi]: (1) Background: The virulence of coronavirus diseases due to viruses like SARS-CoV or MERS-CoV decreases in humid and hot weather. The
[431ksdno]: The coronavirus and the influenza virus have similarities and differences. In order to comprehensively compare them, their genome sequ
[15slu3kk]: SARS CoV-2 (COVID-19) Coronavirus cases are confirmed throughout the world and millions of people are being put into quarantine. A bet
[lj8t52yl]: The origin of the SARS-CoV-2 virus remains enigmatic. It is likely to be a continuum resulting from inevitable mutations and recombina
[0pujch9v]: We don't know if changing seasons will help stem the outbreak, says Michael Le Page
[QREL]:     query_id    doc_id
15          2  01goni72
16          2  03s9spbi
17          2  04awj06g
18          2  04rbtmmi
19          2  0oma7hdu
20          2  0pujch9v
21          2  0vlzwksu
22          2  0y62eqdx
23          2  11pcdnlw
24          2  147yc66p
25          2  15slu3kk
26          2  1bxt21za
27          2  1dq91x2r
28          2  1k3d3o2q
29          2  1llox90t
```

Ranking Evaluation using MRR

Mean Reciprocal Ranking Evaluation metric is calculated as specified by its algorithm. We used this evaluation alongside the other three to find optimal value for the λ parameter. As can be seen, the weighted word2vec can will suggest the relevant document "sooner" than the other models. The result of untrained BERT is in par with the basic proablistic models in the previous problem set which shows the models robustness. The other metrics non the less show the same trend as the MRR, therfor we only show them wihtout specific discriptions.

```
[Unigram_λ=0.1   MRR]: 0.72
[Unigram_λ=0.2   MRR]: 0.74
[Unigram_λ=0.3   MRR]: 0.74
[Unigram_λ=0.4   MRR]: 0.74
[Unigram_λ=0.5   MRR]: 0.74
[Unigram_λ=0.6   MRR]: 0.73
[Unigram_λ=0.7   MRR]: 0.73
[Unigram_λ=0.8   MRR]: 0.73
[Unigram_λ=0.9   MRR]: 0.75
[Bigram, λ_1=0.6, λ_2=0.1   MRR]: 0.65
[Bigram, λ_1=0.6, λ_2=0.2   MRR]: 0.69
[Bigram, λ_1=0.6, λ_2=0.3   MRR]: 0.71
[Bigram, λ_1=0.7, λ_2=0.1   MRR]: 0.66
[Bigram, λ_1=0.7, λ_2=0.2   MRR]: 0.70
[Bigram, λ_1=0.8, λ_2=0.1   MRR]: 0.66
[Bigram, λ_1=0.9, λ_2=0.1   MRR]: 0.66
[Arithmatic W2V   MRR]: 0.69
[Weighted W2V   MRR]: 0.83
[BERT   MRR]: 0.42
```

Ranking Evaluation using MAP

Mean Average Precision of the methods. All the evaluation methods used in this problem set are exactly like the previous one!

```
[Unigram_λ=0.1   MAP]: 0.48
[Unigram_λ=0.2   MAP]: 0.49
[Unigram_λ=0.3   MAP]: 0.48
[Unigram_λ=0.4   MAP]: 0.46
[Unigram_λ=0.5   MAP]: 0.45
[Unigram_λ=0.6   MAP]: 0.45
[Unigram_λ=0.7   MAP]: 0.45
[Unigram_λ=0.8   MAP]: 0.45
[Unigram_λ=0.9   MAP]: 0.45
[Bigram, λ_1=0.6, λ_2=0.1   MAP]: 0.39
[Bigram, λ_1=0.6, λ_2=0.2   MAP]: 0.40
[Bigram, λ_1=0.6, λ_2=0.3   MAP]: 0.40
[Bigram, λ_1=0.7, λ_2=0.1   MAP]: 0.39
[Bigram, λ_1=0.7, λ_2=0.2   MAP]: 0.40
[Bigram, λ_1=0.8, λ_2=0.1   MAP]: 0.39
[Bigram, λ_1=0.9, λ_2=0.1   MAP]: 0.38
[Arithmatic W2V   MAP]: 0.42
[Weighted W2V   MAP]: 0.48
[BERT   MAP]: 0.18
```

Ranking Evaluation using P@K

Percision at K for the k in 5,10 are as follows which confirm both previous results

```
[Unigram_λ=0.1   P@5]: 0.55
[Unigram_λ=0.2   P@5]: 0.56
[Unigram_λ=0.3   P@5]: 0.56
[Unigram_λ=0.4   P@5]: 0.55
[Unigram_λ=0.5   P@5]: 0.55
[Unigram_λ=0.6   P@5]: 0.53
[Unigram_λ=0.7   P@5]: 0.53
[Unigram_λ=0.8   P@5]: 0.53
[Unigram_λ=0.9   P@5]: 0.53
[Bigram, λ_1=0.6, λ_2=0.1   P@5]: 0.44
[Bigram, λ_1=0.6, λ_2=0.2   P@5]: 0.48
[Bigram, λ_1=0.6, λ_2=0.3   P@5]: 0.51
[Bigram, λ_1=0.7, λ_2=0.1   P@5]: 0.45
[Bigram, λ_1=0.7, λ_2=0.2   P@5]: 0.49
[Bigram, λ_1=0.8, λ_2=0.1   P@5]: 0.45
[Bigram, λ_1=0.9, λ_2=0.1   P@5]: 0.44
[Arithmatic W2V   P@5]: 0.49
[Weighted W2V   P@5]: 0.62
[BERT   P@5]: 0.18
```

```
[Unigram_λ=0.1   P@10]: 0.48
[Unigram_λ=0.2   P@10]: 0.49
[Unigram_λ=0.3   P@10]: 0.48
[Unigram_λ=0.4   P@10]: 0.46
[Unigram_λ=0.5   P@10]: 0.45
[Unigram_λ=0.6   P@10]: 0.45
[Unigram_λ=0.7   P@10]: 0.45
[Unigram_λ=0.8   P@10]: 0.45
[Unigram_λ=0.9   P@10]: 0.45
[Bigram, λ_1=0.6, λ_2=0.1   P@10]: 0.39
[Bigram, λ_1=0.6, λ_2=0.2   P@10]: 0.40
[Bigram, λ_1=0.6, λ_2=0.3   P@10]: 0.40
[Bigram, λ_1=0.7, λ_2=0.1   P@10]: 0.39
[Bigram, λ_1=0.7, λ_2=0.2   P@10]: 0.40
[Bigram, λ_1=0.8, λ_2=0.1   P@10]: 0.39
[Bigram, λ_1=0.9, λ_2=0.1   P@10]: 0.38
[Arithmatic W2V   P@10]: 0.42
[Weighted W2V   P@10]: 0.48
[BERT   P@10]: 0.18
```