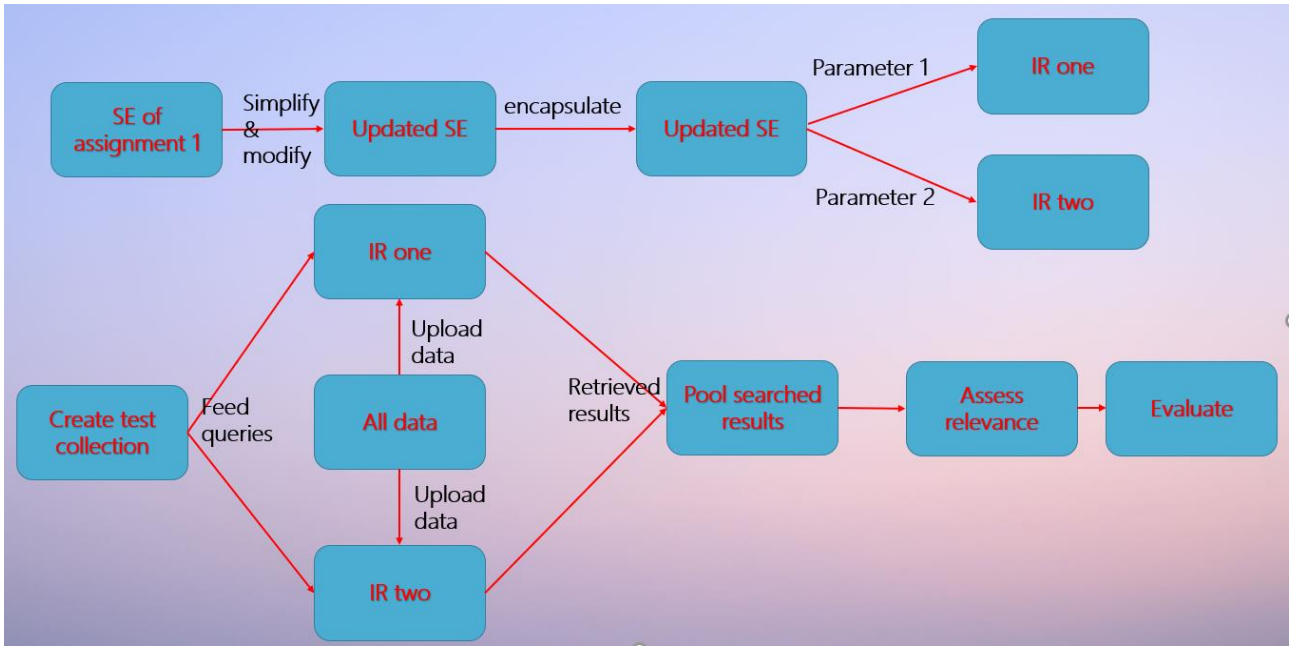
Information Retrieval 2021

Design of the overall architecture (Task 0, preparation)

1) ***Simplify & adjust codes of assignment 1.*** *In assignment 1, a search engine was built, in this assignment, some codes are based on assignment.*
2) ***Encapsulate the codes of assignment 1****, here 2 different IR systems can be created by using different parameters for convenience.*
3) ***Use more data****. In assignment 1, the database is 'Wikipedia Movie Plots', but only 1000 rows were sampled and uploaded to Elasticsearch. While all rows are uploaded in this assignment. Therefore, it may takes much longer to search one query, approximately 3-5 minutes.*
4) ***Create test collection****. 3 queries to search by 2 IR systems.*
5) ***Configure 2 IR systems****. The two IR systems have different parameters when mapping in the Elasticsearch.*
6) ***Pool searched results****. Pool the top-10 retrieved results from 2 IR systems for each quiery.*
7) ***Assess relevance****. Eye-check the pooling results, they are binary judgements, either relevant or non-relevant. It is subjective in some degree.*
8) ***Evaluation.*** *After the relevant ones are judged, then use that as criteria to evaluate the performance of 2 IR systems.*



(architecture, SE -> search engine, IR -> information retrieval)

Test collection (Task 1)

1) ***Information need.*** *Because the database is 'Wikipedia Movie Plots', so built IR systems can provide seach on the features of movies such as origin & year, the plot, (and directors & cast if you like).*
   - *Binary model search, it requires full-match between query and document. For example, where & when the movie produced, namely, origin & year.*
   - *Vector model search, it allows part-match between query and document. For example, the plot information.*
2) ***Created test collection****. Here 3-type searches are used as example for testing.*
   - *Only use binary model seach*
   - *Only use vector model search*

- *Binary + Vector model seach*

3) Details about 3 queries:

Information need	**Query**
Binary Model Search: specific in Origin, years. - retrieve the American movies, - from 1990 to 2010, - about drama	query_1 = {} query_1['yearFrom'] = 1990 query_1['yearTo'] = 2010 query_1['origin'] = 'American' query_1['genre'] = 'drama' query_1['inputText'] = 'american drama'
Vector Model Search: specific in Plot. - retrieve James Bond movies or some adventure mission films, about espionage, spying,	query_2 = {} query_2['yearFrom'] = 1900 # default value, just very early query_2['yearTo'] = 2021 # default value, just very recent query_2['origin'] = '' # no restraint query_2['genre'] = '' # no restraint query_2['inputText'] = 'james bond movies or some adventure mission films, about espionage, spying '
Binary + Vector Model Search: specific in Origin, years, & Plot. - retrieve James Bond movies or some adventure mission films, about espionage, spying, - from 1990 to 2018 - origin: American or British, - genre: drama, action, spy, romance	query_3 = {} query_3['yearFrom'] = 1990 query_3['yearTo'] = 2018 query_3['origin'] = 'American, British' query_3['genre'] = 'drama, action, spy, romance' query_3['inputText'] = 'james bond movies or some adventure mission films, about espionage, spying '

IR systems (Task 2)

1) *The 2 IR systems are also based on assignment 1 but with different configuration parameters. They all have:*
   Indexing stage,
   Tokenization stage,
   Morphological Analysis stage,
   Select keywords stage,
   Search stage.

2) *Difference between the 2 ID systems: the 2 IR systems has different configurations / parameters, namely, they have different mapping ways. One uses stemmer, while another does not.*

IR system 1:

In the code, num=2, using 'my_analyzer02' as analyzer when mapping:

```
"my_analyzer02": {
    "type": "custom",
    "tokenizer": "standard",
    "filter": [ "lowercase", "asciifolding", "english_stop"]
},
```

It only uses to lower case & stop words.

IR system 2:

In the code, num=3, using 'my_analyzer03' as analyzer when mapping:

```
"my_analyzer03": {
      "type": "custom",
      "tokenizer": "standard",
      "filter": [ "lowercase", "asciifolding", "english_stop", "light_english_stemmer"]
 },
```

It uses to lower case & stop words, and stemmer.

Pool method (Task 3)

For each query, record the retrieved top-10 results from the 2 IR systems, and pool them.
For pooling, there are at most 20 documents, but in this assignment,

1) *For query 1, pooling has 10 documents, because the retrieved results from IR 1 & 2 overlap totally, but relative positions have some differences.*
2) *For query 2, pooling has 17 documents, the retrieved results from IR 1 & 2 overlap only 3, and relative positions also have some differences.*
3) *For query 3, pooling has 13 documents, the retrieved results from IR 1 & 2 overlap for most of documents, at the same time, relative positions also have some differences.*

Query	**# different documents**	**Id of the documents retrieve by System 1**	**Id of the documents retrieve by System 2**
query_1 = {} query_1['yearFrom'] = 1990 query_1['yearTo'] = 2010 query_1['origin'] = 'American' query_1['genre'] = 'drama' query_1['inputText'] = 'american drama'	['11970', '12770', '12511', '13648', '14289', '14649', '15428', '14109', '15156', '11866']	['11970', '12770', '12511', '13648', '14289', '14649', '15428', '14109', '15156', '11866']	['11970', '12770', '12511', '13648', '14289', '14649', '15428', '14109', '11866', '15156']
query_2 = {} query_2['yearFrom'] = 1900 # default value, just very early query_2['yearTo'] = 2021 # default value, just very recent query_2['origin'] = '' # no restraint query_2['genre'] = '' # no restraint query_2['inputText'] = 'james bond movies or some adventure mission films, about espionage, spying '	['8169', '3679', '9379', '25476', '24515', '29385', '29364', '15393', '18664', '19672', '5340', '18684', '17151', '10194', '13057', '4794', '15414']	['8169', '3679', '9379', '25476', '24515', '29385', '29364', '15393', '18664', '19672']	['8169', '25476', '5340', '29364', '18684', '17151', '10194', '13057', '4794', '15414']

query_3 = {} query_3['yearFrom'] = 1990 query_3['yearTo'] = 2018 query_3['origin'] = 'American, British' query_3['genre'] = 'drama, action, spy, romance' query_3['inputText'] = 'james bond movies or some adventure mission films, about espionage, spying '	['14852', '15067', '20612', '15507', '21001', '20801', '20738', '20663', '14910', '14610', '15939', '12338', '17281']	['14852', '15067', '20612', '15507', '21001', '20801', '20738', '20663', '14910', '14610']	['14852', '15067', '20663', '14610', '15939', '12338', '17281', '20612', '15507', '21001']

Relevance assessments (Task 4)

Here just eye-check every document in pooling for each query, read the document details, and then give a binary justification, namely, relevant or non-relevant. But those judgements may be subjective in some degree. (human judgement may take quite long time, for at most 60 documents.)
Relevance criteria:

Query	**ID of relevant documents**
query_1 = {} query_1['yearFrom'] = 1990 query_1['yearTo'] = 2010 query_1['origin'] = 'American' query_1['genre'] = 'drama' query_1['inputText'] = 'american drama'	['11970', '12770', '12511', '13648', '14289', '14649', '15428', '14109', '15156', '11866']
query_2 = {} query_2['yearFrom'] = 1900 # default value, just very early query_2['yearTo'] = 2021 # default value, just very recent query_2['origin'] = '' # no restraint query_2['genre'] = '' # no restraint query_2['inputText'] = 'james bond movies or some adventure mission films, about espionage, spying '	['8169', '3679', '29364', '18664', '19672', '5340', '18684', '13057']
query_3 = {} query_3['yearFrom'] = 1990 query_3['yearTo'] = 2018 query_3['origin'] = 'American, British' query_3['genre'] = 'drama, action, spy, romance' query_3['inputText'] = 'james bond movies or some adventure mission films, about espionage, spying '	['14852', '20612', '15507', '21001', '20801', '20738', '20663', '14910', '15939', '12338']

Evaluation (Task 5)

For each query:

1) *Form a criteria list that contains the relevant indexes, after eye-checking / human judgement.*
2) *Get the top-5 retrieved results (indexes) from IR 1 & 2 as two lists.*
3) *Iterate every element in the list,*
   *define a ratio variable **P5**,*
   *if the element is contained in criteria, **P5**+1,*
   *finally, **P5** divided by 5, that is the* **P@5**.

   *define a ratio variable **R5**,*
   *if the element is contained in criteria, **R5**+1,*
   *finally, **R5** divided by the length of criteria list, that is the* **R@5**.

```
def evaluation(query_num, col):
    # eye-checking results as the criteria
    criteria = assess_relevance(query_num)

    # P@5
    p5 = {}
    for i in col:
        ir = col[i][:5] # get the top 5
        tmp = []
        s = 0
        count = 0
        for it in ir:
            count = count + 1
            if it in criteria:
                s = s + 1
                tmp.append(s/count)
            else:
                tmp.append(s/count)
        p5[i] = tmp
```

```
    # R@5
    r5 = {}
    L = len(criteria)
    for i in col:
        ir = col[i][:5] # get the top 5
        tmp = []
        s = 0
        for it in ir:
            if it in criteria:
                s = s + 1
                tmp.append(s/L)
            else:
                tmp.append(s/L)
        r5[i] = tmp

    return p5, r5
```

(code for evaluation part, method evaluation)

But in this assignment, **P@1 - P@5** *and* **R@1 - R@5** *are all calculated with the above illustrated thoughts.*

```
2 IR systems searching with query 1:
P@5     {2: [1.0, 1.0, 1.0, 1.0, 1.0], 3: [1.0, 1.0, 1.0, 1.0, 1.0]}
R@5     {2: [0.1, 0.2, 0.3, 0.4, 0.5], 3: [0.1, 0.2, 0.3, 0.4, 0.5]}
2 IR systems searching with query 2:
P@5     {2: [1.0, 1.0, 0.6666666666666666, 0.5, 0.4], 3: [1.0, 0.5, 0.6666666666666666, 0.75, 0.8]}
R@5     {2: [0.125, 0.25, 0.25, 0.25, 0.25], 3: [0.125, 0.125, 0.25, 0.375, 0.5]}
2 IR systems searching with query 3:
P@5     {2: [1.0, 0.5, 0.6666666666666666, 0.75, 0.8], 3: [1.0, 0.5, 0.6666666666666666, 0.5, 0.6]}
R@5     {2: [0.1, 0.1, 0.2, 0.3, 0.4], 3: [0.1, 0.1, 0.2, 0.2, 0.3]}
```

(running screenshot)

Illustrations about the output:

For query 1	**P@1**	**P@2**	**P@3**	**P@4**	**R@1**	**R@2**	**R@3**	**R@4**
IR 1	*1.0*	*1.0*	*1.0*	*1.0*	*0.1*	*0.2*	*0.3*	*0.4*
IR 2	*1.0*	*1.0*	*1.0*	*1.0*	*0.1*	*0.2*	*0.3*	*0.5*

For query 2	**P@1**	**P@2**	**P@3**	**P@4**	**R@1**	**R@2**	**R@3**	**R@4**
IR 1	*1.0*	*1.0*	*0.67*	*0.5*	*0.125*	*0.25*	*0.25*	*0.25*
IR 2	*1.0*	*0.5*	*0.67*	*0.75*	*0.125*	*0.125*	*0.25*	*0.375*
For query 3	**P@1**	**P@2**	**P@3**	**P@4**	**R@1**	**R@2**	**R@3**	**R@4**
IR 1	*1.0*	*0.5*	*0.67*	*0.75*	*0.1*	*0.1*	*0.2*	*0.3*
IR 2	*1.0*	*0.5*	*0.67*	*0.5*	*0.1*	*0.1*	*0.2*	*0.2*

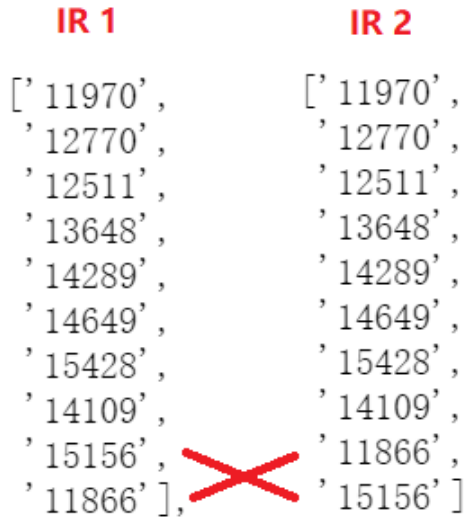
Measure of **P@5** & **R@5:**

	System 1		System 2	
	P@5	**R@5**	**P@5**	**R@5**
Q1	1	0.5	1	0.5
Q2	0.4	0.25	0.8	0.5
Q3	0.8	0.4	0.6	0.3

Discussion & analysis:

For the performance records above, it is obvious:

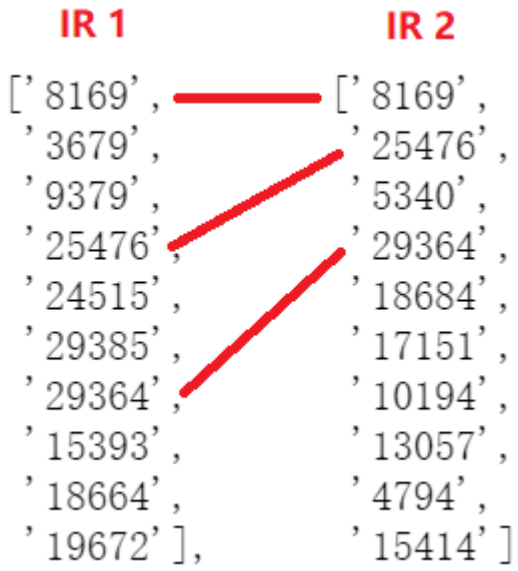
1) For query 1 that is binary model search, IR 1 and IR 2 perform almost the same, they retrieve the same indexes (documents) with little difference in the relative rankings for the last 2 of top-10 retrieval.



(query 1, comparison between IR 1 & 2)

Analysis: because the searches are quite simple and straightforward, only limiting the origin & year & genre for movies retrievals. So, both IR 1 & 2 perform well and almost the same.

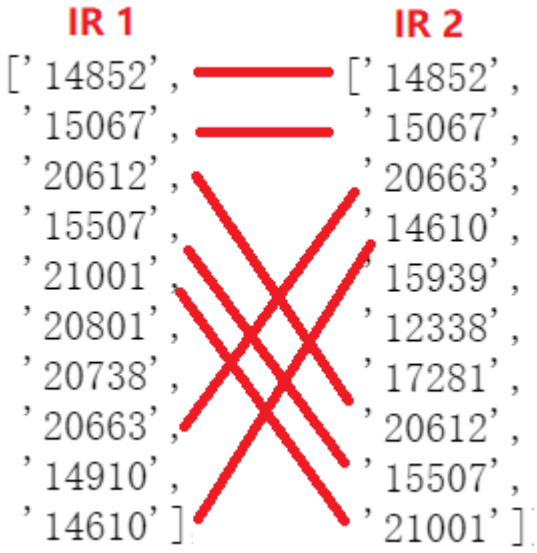
2) For query 2 that is vector model search, IR 1 and IR 2 perform much differently, their retrievals overlap only 3 indexes, and for the top-5, IR 2 returns 3 relevant while IR 1 only 2.



(query 2, comparison between IR 1 & 2)

Analysis: this query is very difficult because it only provides short and vague query, also allows part-match. For IR 1 without stemmer, if words in documents have different tense or morphology from that in the query, those documents likely are missed. While for IR 2, because it uses stemmer, naturally, its performance will be better.

3) For query 3 that is binary + vector model search, for the top-10 retrievals, IR 1 and IR 2 share 7 same indexes (documents). While for the top-5 performance, IR 1 is better that IR 2. (see the P@5 & R@5 table)



(query 3, comparison between IR 1 & 2)

Analysis: query 3 is the combination of query 1 & 2.
IR 1 without stemmer retrieved 4 relevant documents for top-5, while IR 2 with stemmer only 3. Theoretically, IR 2 with stemmer should perform better than IR 1 without stemmer, their other configurations are the same. The reason may be the query is quite straightforward and simple. If use more complex queries, the results may be quite different.

Summary:
In this assignment, 2 ID systems with different configurations, one using stemmer while the other not, performs differently on 3 queries. Their performances are recorded and analyzed. Here, P@5 & R@5 are taken as evaluation measures.

Generally, using stemmer and something morphological analysis performs better than not using, but not definitely.

Future work:
in this assignment, the semantic meanings of query and documents are not considered, which is trending at present such as NLP (natural language processing). If convert query and documents into vectors (not simple one-hot encoding), namely, word embedding, it is believed that lots of performance will be improved.

Reference:
[1] Elasticsearch, python, documents. (n.d.). Elasticsearch. Retrieved April 8, 2021, from https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-custom-analyzer.html