COL764

Information Retrieval and Web Search
Conversational Search

Introduction

Conversational search is one of the ultimate goals of information search. A lot of recent searches are based on formation of a valid query from a given conversation and then ranking the given documents accordingly in order to match the conversation's information needs. Conversation is made up of various kinds of dialogues, which includes questions about a topic, some assertions on it and some counter questions on it.
So, while looking at the Conversational Search, we concluded the various points ::
1. A conversational question has a heavy dependency on the dialogues in history of that conversation.
2. Simply appending all the dialogues as text will not help much because the redundant words or expression of that conversation will constitute a large fraction of it.

Project Brief

We tried finding some methods to achieve the task of conversational search which is defined as

- A dataset of some conversations will be given to us, in which the recent question of that conversation along with the history of the conversation is provided to us.
- A corpus of the passages from wikipedia is given out of which we intend to retrieve the top 20 passages in which the user can find the probable answer of that conversational question.

Dataset Used

PARAGRAPHS -   A total of around 5 million passages constitute the dataset, each passage has a title, an id.

Link to it (all_blocks.txt.gz) :- https://ciir.cs.umass.edu/downloads/ORConvQA

all_blocks.txt 〉 No Selection


```
{'text': 'Hovertrain A hovertrain is a type of high-speed train that replaces conventional steel wheels with hovercraft lift pads, and the conventional railway bed with a paved road-like surface, known as the "track" or "guideway". The concept aims to eliminate rolling resistance and allow very high performance, while also simplifying the infrastructure needed to lay new lines. Hovertrains were seen as a relatively low-risk and low-cost way to develop high-speed inter-city train service, in an era when conventional rail seemed stuck to speeds around or less. By the late 1960s, major development efforts were underway in France, the UK and the USA. While they were being developed, British Rail was running an extensive study of the problems being seen at high speeds on conventional rails. This led to a series of new high-speed train designs in the 1970s, starting with their own APT. Although the hovertrains still had reduced infrastructure costs compared to the APT and similar designs like the TGV, in practice this was offset by their need for entirely new lines. Conventional wheeled trains could run at low speed on existing lines, greatly reducing capital expenditures in urban areas. Interest in hovertrains waned, and major development had ended by the mid-1970s. Hovertrains were also developed for smaller systems, including personal rapid transit systems that were a hot topic in the late 1960s and early 1970s. In this role their ability to float over small imperfections and debris on the "rails" was a practical advantage, although it competed with the maglev concept that had the same advantages. The only hovertain to see commercial service was the Otis Hovair system. Originally developed at General Motors as an automated guideway transit system, GM was forced to divest the design as part of an antitrust ruling.', 'title': 'Hovertrain', 'aid': '25747583', 'bid': 0, 'id': '25747583@0'}
{'text': 'The design eventually ended up at Otis Elevator who later replaced its linear motor with a cable pull and sold the resulting design for people mover installations all over the world. Hovertrain is a generic term, and the vehicles are more commonly referred to by their project names where they were developed. In the UK they are known as tracked hovercraft, in the US they are tracked air-cushion vehicles. The hovertrain was originally developed by Jean Bertin (1917–1975) in France, where they were marketed as the Aérotrain (1965–1977), before it was eventually abandoned by the French government. It was noticed early on that the energy needed to lift a hovercraft was directly related to the smoothness of the surface it traveled on. This was not surprising; the air trapped under the hovercraft\'s skirt will remain there except where it leaks out around the bottom of the skirt where it contacts the ground – if this interface is smooth, the amount of leaked air will be low. What was surprising was that the amount of energy lost through this process could be lower than steel wheeled vehicles, at least at high speeds. At high speeds, trains suffer from a form of instability known as "hunting oscillation" that forces the flanges on the sides of the wheels to hit the sides of the rails, as if they were rounding a tight bend. At speeds of or over, the frequency of these hits increased to the point where they became a major form of drag, dramatically increasing rolling resistance and potentially causing a derailment. That meant that for travel above some critical speed, a hovercraft could be more efficient than a wheeled vehicle of the same weight. Better yet, such a vehicle would also retain all of the positive qualities of a hovercraft.', 'title': 'Hovertrain', 'aid': '25747583', 'bid': 1, 'id': '25747583@1'}
{'text': 'Small imperfections in the surface would have no effect on the ride quality, so the complexity of the suspension system could be reduced. Additionally, since the load is spread out over the surface of the lifting pads, often the entire underside of the vehicle, the pressure on the running surface is greatly reduced – about the pressure of a train wheel, about of the pressure of a tire on a road. These two properties meant that the running surface could be considerably simpler than the surface needed to support the same vehicle on wheels; hovertrains could be supported on surfaces similar to existing light-duty roadways, instead of the much more complex and expensive railbeds needed for conventional trains. This could dramatically reduce infrastructure capital costs of building new lines and offer a path to widespread use of high-speed trains. One of the earliest hovertrain concepts predates hovercraft by decades; in the early 1930s Andrew Kucher, an engineer at Ford, came up with the idea of using compressed air to provide lift as a form of lubrication. This led to the Levapad concept, where compressed air was blown out of small metal disks, shaped much like a poppet valve. The Levapad required extremely flat surfaces to work on, either metal plates, or as originally intended, the very smooth concrete of a factory floor. Kucher eventually became VP in charge of the Ford Scientific Laboratory, continuing development of the Levapad concept throughout. It does not appear any effort was put into vehicle use until the 1950s, when several efforts used Levapad-like arrangements running on conventional rails as a way to avoid the hunting problems and provide high-speed service. A 1958 article in "Modern Mechanix" is one of the first popular introductions of the Levapad concept.', 'title': 'Hovertrain', 'aid': '25747583', 'bid': 2, 'id': '25747583@2'}
{'text': 'The article focuses on cars, based on Ford\'s prototype "Glideair" vehicle, but quotes Kucher noting "We look upon Glideair as a new form of high-speed land transportation, probably in the field of rail surface travel, for fast trips of distances of up to about 1,000 miles [1,600 km]". A 1960 "Popular Mechanics" article notes a number of different groups proposing a hovertrain concept. What was lacking from all of them was a suitable way to move the vehicles forward – since the whole idea of the hovertrain concept was to eliminate any physical contact with the running surface, "especially" wheels, some sort of contact-less thrust would have to be provided. There were various proposals using air ducted from the lift fans, propeller, or even jet engines, but none of these could approach the efficiency of an electric motor powering a wheel. At about the same time, Eric Laithwaite was building the first practical linear induction motors (LIMs), which, prior to his efforts, had been limited to "toy" systems. A LIM can be built in several different ways, but in its simplest form it consists of an active portion on the vehicle corresponding to the windings on a conventional motor, and a metal plate on the tracks acting as the stator. When the windings are energized, the magnetic field they produce causes an opposite field to be induced in the plate. There is a short delay between field and induced field due to hysteresis. By carefully timing the energizing of the windings, the fields in the windings and "reaction rail" will be slightly offset due to the hysteresis. That offset results in a net thrust along the reaction rail, allowing the LIM to pull itself along the rail without any physical contact.', 'title': 'Hovertrain', 'aid': '25747583', 'bid': 3, 'id': '25747583@3'}
{'text': 'The LIM concept sparked considerable interest in the transportation world, as it offered a way to make an electric motor with no moving parts and no physical contact, which could greatly reduce maintenance needs. Laithwaite suggested that the LIM would be a perfect fit for high speed transport, and built a model consisting of a chair mounted on a four-wheeled chassis on rails with a LIM rail running down the middle. After successful demonstrations, he convinced British Rail (BR) to invest in some experimental work using a LIM to power a train on rails using small lift pads similar to the Levipad system for suspension. As the various hovertrain systems were developing, a major energy use issue cropped up. Hovercraft generate lift by providing pressure, as opposed to generating lift due to the momentum of air flowing over an airfoil. The pressure of the air required is a function of the vehicle weight and the size of the lift pad, essentially a measure of overall vehicle density. A non-moving vehicle only loses this air due to leakage around the pads, which can be very low depending on the relative pressure between the pad and the outside atmosphere, and further reduced by introducing a "skirt" to close the gap between the pad and
```


__Format__

Contains a list of dictionary where each dictionary contains the following fields :-

1. "text" : Contains the text of the passage
2. "title" : This field represents the title of the passage, i.e., the context of it
3. "aid" : This field is a number which has the same number for all the passage having the same title
4. "bid" : This number represents the numbering of the passage of the same title
5. "id" : It is a string which is the unique id of every passage which is basically the concatenation of the aid and the bid with "@" in between them.

Dialogues - The dataset contains 5644 dialogue sets (complete conversation) and 40,527 questions which constitute all the questions in a dialogue set. (CANARD Dataset)

This data set

Link to it :- https://sites.google.com/view/qanta/projects/canard


```json
[{ "History": [
            "Johnny Unitas",
            "1964 MVP season"
        ],
        "QuAC_dialog_id": "C_2ba58216460d43aa986fc0e897537239_0",
        "Question": "what team did unitas play for",
        "Question_no": 1,
        "Rewrite": "what team did Johnny Unitas play for?"
    },
    {
        "History": [
            "Johnny Unitas",
            "1964 MVP season",
            "what team did unitas play for",
            "The Colts"
        ],
        "QuAC_dialog_id": "C_2ba58216460d43aa986fc0e897537239_0",
        "Question": "how many games did the colts win",
        "Question_no": 2,
        "Rewrite": "how many games did the colts win"
    },
    {
        "History": [
            "Johnny Unitas",
            "1964 MVP season",
            "what team did unitas play for",
            "The Colts",
            "how many games did the colts win",
            "the Colts ran off 10 straight victories to finish with a 12-2 record."
        ],
        "QuAC_dialog_id": "C_2ba58216460d43aa986fc0e897537239_0",
        "Question": "who did they play in the playoffs",
        "Question_no": 3,
        "Rewrite": "who did the Colts play in the playoffs?"
    },
    {
        "History": [
            "Johnny Unitas",
            "1964 MVP season",
            "what team did unitas play for",
            "The Colts",
            "how many games did the colts win",
            "the Colts ran off 10 straight victories to finish with a 12-2 record.",
            "who did they play in the playoffs",
            "Cleveland Browns"
        ],
        "QuAC_dialog_id": "C_2ba58216460d43aa986fc0e897537239_0",
        "Question": "did they win the super bowl",
        "Question_no": 4,
        "Rewrite": "did the Colts win the super bowl?"
    },
    {
        "History": [
            "Johnny Unitas",
            "1964 MVP season",
            "what team did unitas play for",
            "The Colts",
            "how many games did the colts win",
            "the Colts ran off 10 straight victories to finish with a 12-2 record.",
            "who did they play in the playoffs",
            "Cleveland Browns",
            "did they win the super bowl",
            "losing 27-0."
        ],
        "QuAC_dialog_id": "C_2ba58216460d43aa986fc0e897537239_0",
        "Question": "who did they play in the super bowl",
        "Question_no": 5,
        "Rewrite": "who did the Colts play in the super bowl?"
    },
    {
        "History": [
            "Johnny Unitas",
            "1964 MVP season",
            "what team did unitas play for",
            "The Colts",
            "how many games did the colts win",
            "the Colts ran off 10 straight victories to finish with a 12-2 record.",
            "who did they play in the playoffs",
            "Cleveland Browns",
            "did they win the super bowl",
            "losing 27-0.",
            "who did they play in the super bowl",
            "the Packers."
        ],
        "QuAC_dialog_id": "C_2ba58216460d43aa986fc0e897537239_0",
        "Question": "what were unitas stats",
        "Question_no": 6,
        "Rewrite": "what were Johnny Unitas stats?"
    },
    {
        "History": [
            "Mark Taylor (cricketer)",
            "Early years"
        ],
        "QuAC_dialog_id": "C_ae269bdc0d524b599736eb69a322d5b1_1",
        "Question": "Where was he born?",
        "Question_no": 1,
        "Rewrite": "Where was Mark Taylor born?"
    },
    {
        "History": [
            "Mark Taylor (cricketer)",
```


Format

Contains a list of dictionary where each dictionary contains the following fields :-

1. History : It is a list containing all the dialogues in the form of a string which were said before the recent question in the conversation.
2. QuAC_dialog_id : It is an id unique to every question in every conversation.
3. Question : It is the most recent dialog (which is a question) of the conversation.
4. Question_no : It is the numbering of the question in the same dialog set.
5. Rewrite : Rewrite is the ideal question written by a human which would produce the correct result of the question in the conversation.

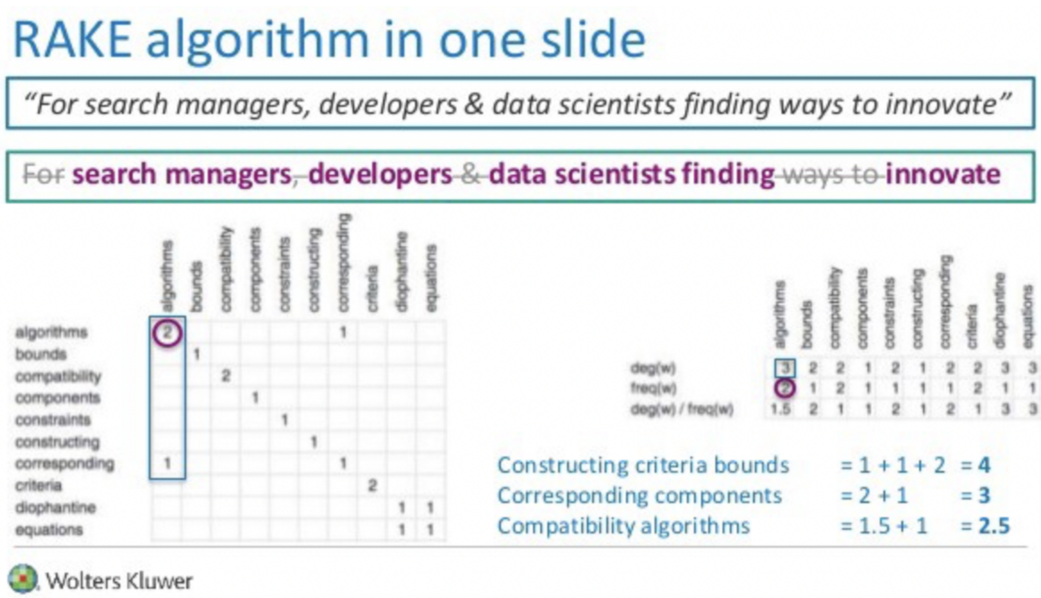
Limitations of the dataset

The first question of the conversation is the main question giving the idea about the context of the conversation, so we cannot ignore the first question while formulating any new queries to search the relevant passages and in real life we may not get these types of conversations every time.

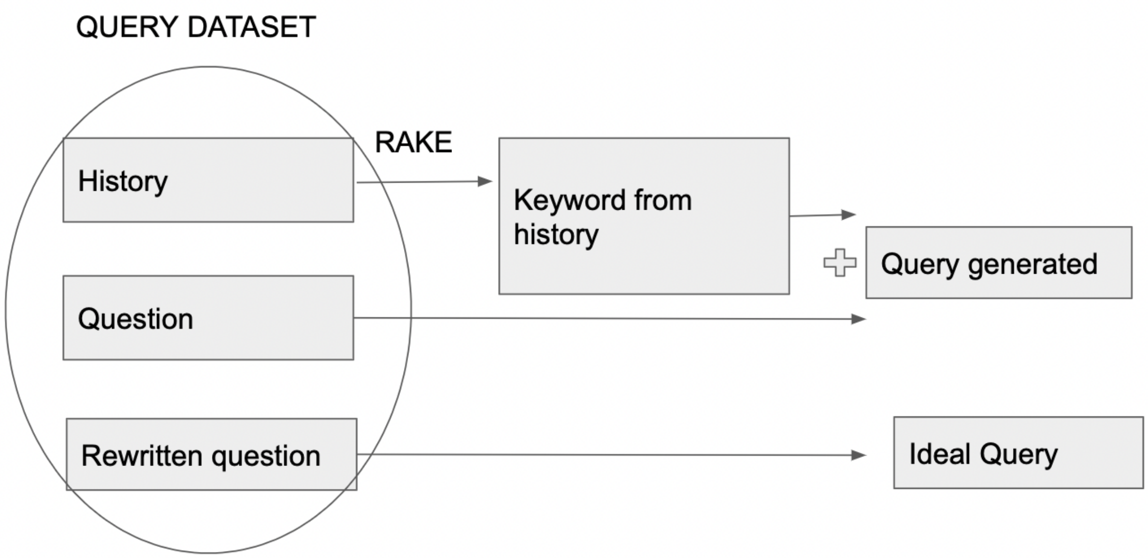
Keyword Extraction

The first goal of our project was to extract the keywords from a given conversational snippet. It would be very useful in conversational search, as it only extracts the important data from the conversation which can later be used to retrieve and rank the relevant documents. We used to strategies to convert a conversational snippet to a search-able query:

- **RAKE**: Rapid Automatic Keyword Extraction(RAKE) is a Domain-Independent keyword extraction algorithm in Natural Language Processing. Rake firstly determines the main contents of the text given to it by eliminating the stopwords and delimiters. After that each word is given a score on the basis of degree and frequency of the word. The words and phrases are then ranked according to these scores and ranked keywords are returned. Yake keyword extractor was also tried and then Rake was chosen to go forward with.



As shown above, the queries we have are a combination of history, and a last conversational question. We apply RAKE keyword extraction on the history content, to get the main highlight of the question, and append this to the main question, i.e., the last conversational question. This generated query is passed to the BM25 model, crf_suite model, and log-logistic model.



- **Embedding using BERT**: Queries and conversations were encoded using a pre trained bert encoder. This is discussed in more details in the bert model explanation section.

Retrieval and Ranking:

- **Using CRF MACHINE LEARNING MODELS** :

  The problem of document ranking has been modelled to a classification problem. The ideal top documents for a query are retrieved by applying the BM25 model using n=1, i.e., we are generating the document that is most relevant to a query. The query passed to the BM25 model is the Rewritten question which is the ideal query.

```
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    c1=0.1,
    c2=0.1,
    max_iterations=100,
    all_possible_transitions=True
)

crf.fit(X_train, y_train)
labels = list(crf.classes_)

y_pred = crf.predict(X_train)

print("Train accuracy:", metrics.flat_f1_score(y_train, y_pred,
average='weighted', labels=labels))

y_pred = crf.predict(X_test)
print("Test accuracy:", metrics.flat_f1_score(y_test, y_pred,
average='weighted', labels=labels))
```

  The model is trained using the subset ideal (rewritten questions) as the training data and the corresponding labels for each query is the top document. Model training is done using sklearn_crfsuite model. Now, all such documents generated, (one top document for each query) are modelled as the labels under which the queries should be classified.

  The trained model is then run on the test dataset, i.e., the generated queries, and the test accuracy observed is 80.17%.

- **BM25 MODEL** :

  We generated results on the three types of queries:
    - Ideal query: The rewritten question for each conversation.
    - Without history: The question for each conversation.
    - With history: The question for each conversation appended with the top keyword extracted from the history list of each conversation. The keyword was extracted using RAKE.

The results when history was taken into account while generating the query gave much better results than the one in which only the last question was considered. It is because the conversation is highly dependent on its history.

Top 20 documents are retrieved for each generated query using the pre-trained BM25Okapi model. The ideal documents are retrieved by the ideal queries, i.e., the rewritten queries, and the test documents are retrieved using the query formulated as given in the figure above. This model gave us an MRR score of 0.47 and average F-1 score of 0.21.

- **LOG - LOGISTIC RETRIEVAL MODEL :**

  Need for it :- Apply the Sentence Transformer on all the passages (5 million) for encoding them into vectors is and storing those vectors is computationally very heavy and impossible for us,

  Task accomplished by it :- We used the generated query of the previous model to generate top-100 relevant passages using the log-logistic retrieval method and now those are the subsets of the passages on which we will apply the BERT model to find their relevance for the query.

  Method :-

$$\mathrm{RSV}(Q, D) = \sum_{w \in Q} \mathrm{count}(w, Q) \log\left(\frac{\mathrm{tf}(w, D) + \lambda_w}{\lambda_w}\right), \tag{1}$$

where $\mathrm{tf}(w, D) = \mathrm{count}(w, D) \times \log(1 + c\frac{\mathrm{avdl}}{|D|})$, c is a free parameter, avdl is average document length in the collection and $\lambda_w = \frac{N_w}{N}$ where N_w is the number of documents containing w and N is the number of documents in the collection.

The model provided an MRR score of 0.65 while testing it on the dataset of 10000 docs out of which top-10 were to be retrieved.

- **BERT (SENTENCE TRANSFORMER MODEL) :-**

Original query :- Let the original question of the conversation be q_k, having history $q_1, q_2, q_3, \ldots q_{k-1}$

Input query :- In the model we pass on the concatenation of the queries in the following way :-

[CLS] q_1 [SEP] q_{k-w} [SEP] · · · [SEP] q_{k-1}[SEP] q_k [SEP]

Here [CLS] and [SEP] are the tokens of BERT and w is the window taken by us which refrains us from considering the complete conversation, as dialogues at the very starting of any conversation may have been answered and has less relevance with the question at the end. The first question is still appended to the reformulated query as it contains the broader context of the conversation.

Original passage :- The passage is a string containing various sentences separated by full stop.

Input passage :- The input to the BERT model cannot be much longer for it to work properly, and since the dataset which we contain have the lines having the same context in a passage, so we applied averaging on the similarity score of all the sentences of a passage with respect to the query.

That is, is a paragraph p = $s_1 s_2 s_3 s_4 \ldots s_n$ Where s_i is a sentence is the passage, then we had chosen two options to find the similarity,

1. Here the similarity between a query and a passage is the average of the cosine similarities between the query embeddings and all the sentences embeddings in the passage, i.e.,

Similarity(q ,p) = (Σ CosineSimilarity(query_embedding , s_i_embedding)) / n

2. Here the similarity between a query and a passage is the maximum of all the cosine similarities between the query embeddings and all the sentences embeddings in the passage, i.e.,

Similarity(q ,p) = Max (CosineSimilarity(query_embedding , s_i_embedding))

The first similarity metric gave better results than the second and hence we carried the first metric forward.

We deployed the pretrained BERT models for encoding the re-formulated query and the sentences in the passage which is the "bert-base-nli-mean-tokens" .

- **<u>OTHER MODELS TRIED</u> :-**

  We tried to deploy the **gensim** model for finding the doc2vec which is the vector embedding of the passages, by training this model on our dataset.

  This is a classification model which first finds the classes of the documents present and then groups them by their classes and then proceeds further in classifying a new document under the given classes.

  https://towardsdatascience.com/detecting-document-similarity-with-doc2vec-f8289a9a7db7

  We worked on deploying this model but it proved to be of no use as the dataset of the passages which we have contains the number of titles and the number of passages in the ratio ⅗, which clearly shows that for a lot of titles we don't even have 2 documents, so this model provided no good results.

EVALUATION METRICS USED

- **MRR** :- rank(i) for each query is the top rank at which an ideal document occurs. The list of ideal documents is generated by applying the BM25 model on the ideal query.

$$\mathrm{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\mathrm{rank}_i}.$$

- **Precision** :- tp/(tp+fp)
- **Recall** :- tp/(tp+fn)

- **F1 score** :- tp/(tp+(fp+fn)/2)

where,
tp = Number of documents which are in top-20 in our ideal ranking and which are in top-20 in the ranking of our model,,
fp = Number of documents which are not in top-20 in the ideal ranking but which are in top-20 in the ranking of our model ,
fn = Number of documents which are in top-20 in the ideal ranking but which are not in top-20 in the ranking of our model.

Files used (submitted on github)

Link: https://github.com/skshruti/COL764Project

- **Bm25.py**: This file contains the implementation of the BM25 model using BM25Okapi from rank_bm25.
- **Loglogistic.py:** This file contains the implementation of the log logistic model.
- **Bert.py:** This file contains the implementation of ranking using the pre-trained bert-base-nli-mean-tokens model.
- **Crf.py:** It contains a classification model trained by using our training data using sklearn_crfsuite.
- **Evaluate.py:** This contains the evaluation function which generates MRR, F1 scores, precision and recall given a ranking and an ideal ranking for a set of queries.
- **Query.py:** This file contains the code to compare query taken with history and without history.

Results:

	MRR	**F1 score**
BM25	0.47	0.28
BERT	0.31	0.18
Log logistic	0.41	0.24

Conclusions:

In this project, we have applied and compared various ways of retrieving and ranking documents and extracting keywords from them. The sklearn_crfsuite was a good model, but not the best for our problem as we had to reformulate our document retrieving model to a query classification model. Bert model can also be trained on our dataset and then fine tuned according to our dataset specifications to get better results.

References:

- Contextualized Query Embeddings for Conversational Search arXiv:2104.08707
- Chen Qu, Liu Yang, Cen Chen, Minghui Qiu, W. Bruce Croft, and Mohit Iyyer. 2020. Open-Retrieval Con- versational Question Answering. In *Proc. SIGIR*. https://arxiv.org/pdf/2005.11364.pdf
- https://towardsdatascience.com/detecting-document-similarity-with-doc2vec-f8289a9a7db7
- http://ceur-ws.org/Vol-2036/T3-1.pdf
- RAKE: Rapid Automatic Keyword Extraction Algorithmhttps://medium.datadriveninvestor.com › rake-rapid-aut...
- http://www.treccast.ai/
- https://link.springer.com/chapter/10.1007/978-3-030-45439-5_30
- https://www.analyticsvidhya.com/blog/2021/06/why-and-how-to-use-bert-for-nlp-text-classification/
- https://rcd2020firetask.github.io/RCD2020FIRETASK/