Information Retrieval CW3

Search Engine Implementation

Queen Mary
University of London

BACKGROUND

Aim

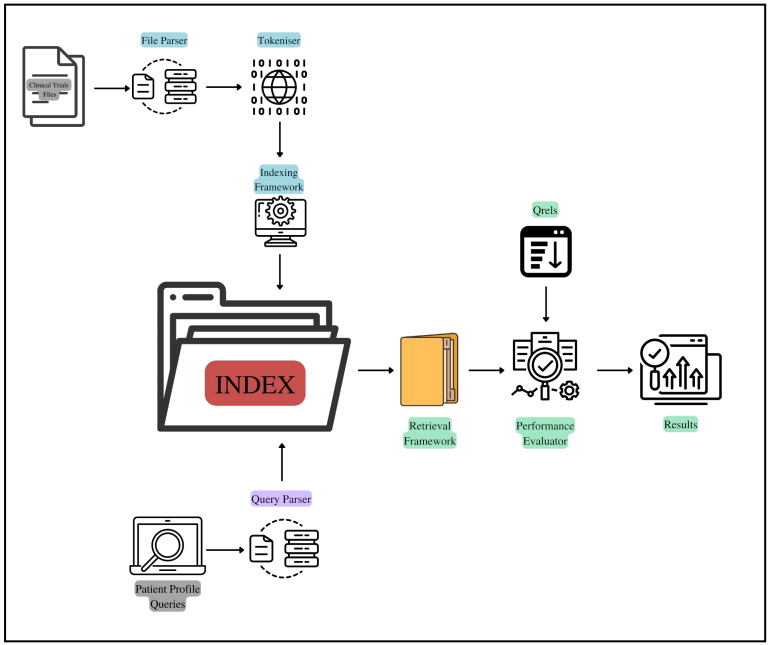
A large amount of clinical trials do not meet their requirement criteria for patient uptakes due to the lengthy unstructured documents and large amount of time it takes for researchers to review trials physically. Therefore, this project establishes a search engine that utilises a patient-to-trial paradigm to retrieve relevant trials given a patient profile.

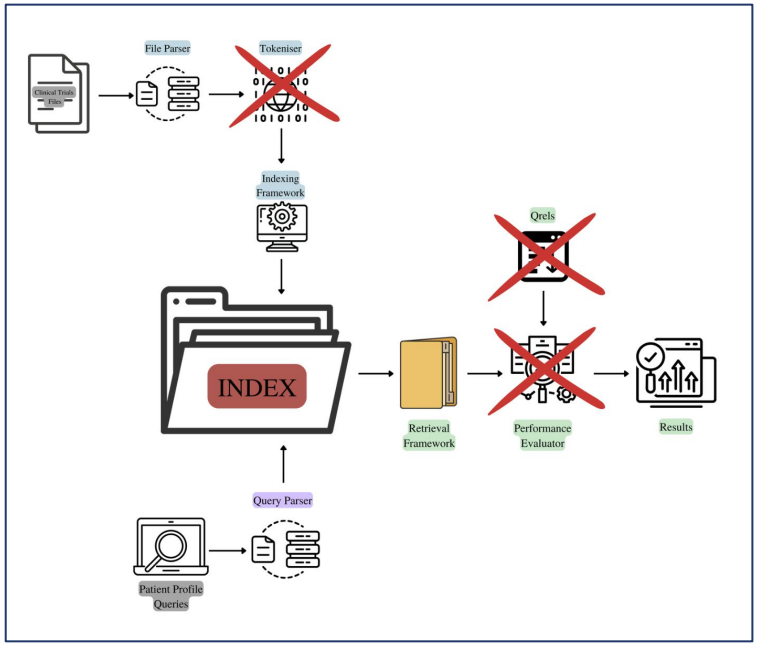
Dataset

The project utilised the Clinical Trials 2021 TREC:

- **Database**: ClinicalTrials 2021 part 1 - NCT0075 to NCT0099 (16k clinical trials)
- **Query**: Patient profiles given in the ClinicalTrials 2021 (75 patient profiles)
- **Relevance**: The Qrels2021 document (35k ground truths for various files)

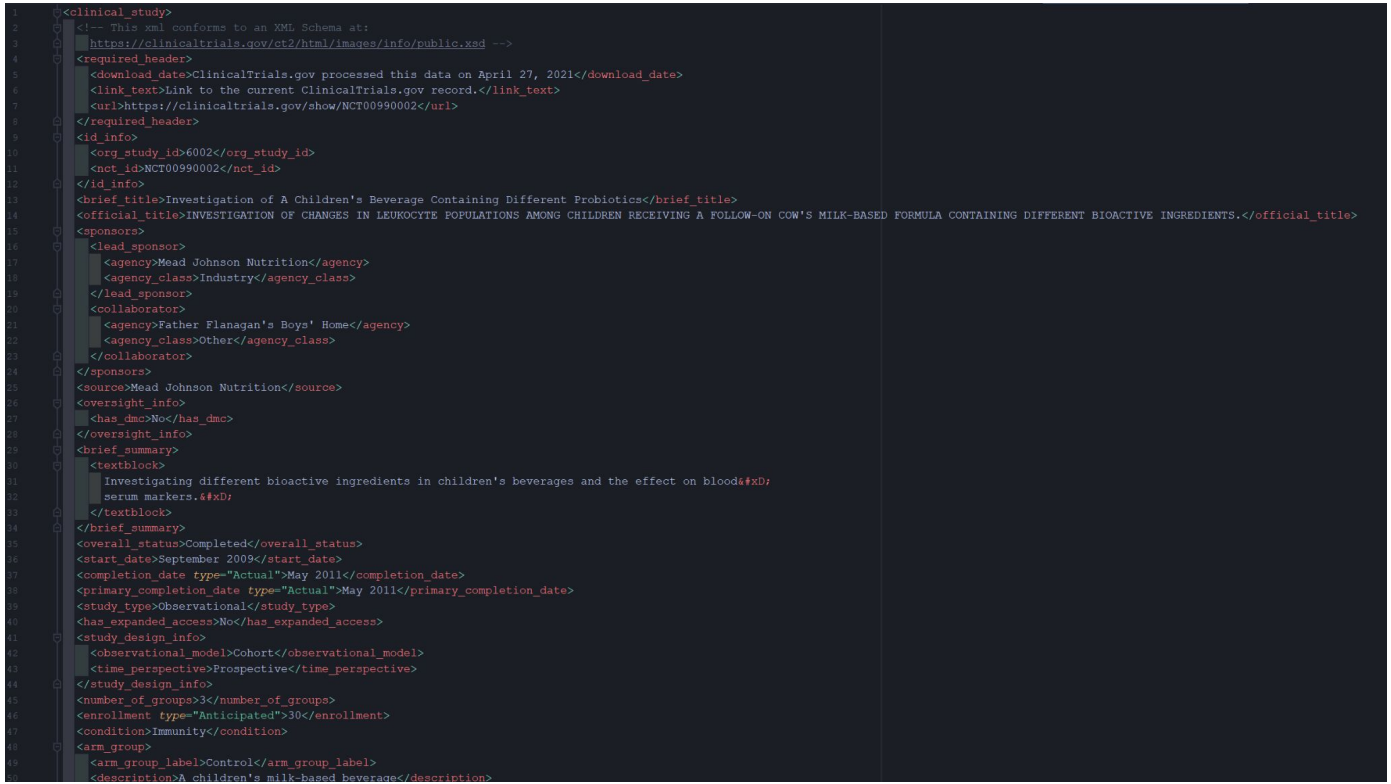
MODIFICATIONS OF PROPOSED PLAN





- No tokenization of files - ElasticSearch automatically tokenizes the database
- Relevance and Performance Evaluation was conducted after results were gathered

HTML FILES

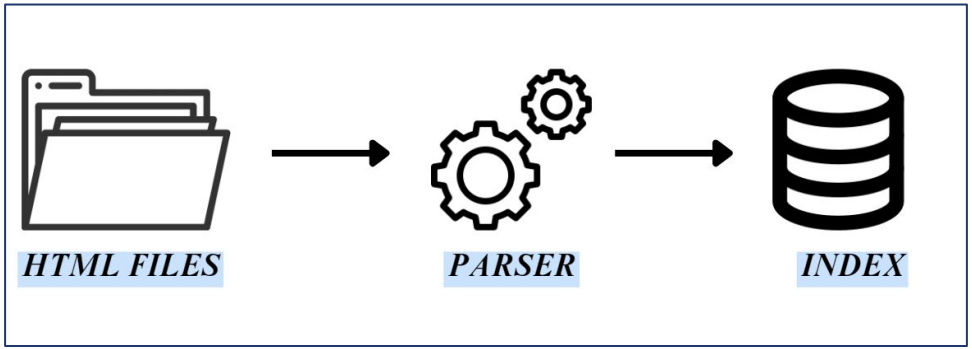


```
<clinical_study>
  <!-- This xml conforms to an XML Schema at:
  https://clinicaltrials.gov/ct2/html/images/info/public.xsd -->
  <required_header>
    <download_date>ClinicalTrials.gov processed this data on April 27, 2021</download_date>
    <link_text>Link to the current ClinicalTrials.gov record.</link_text>
    <url>https://clinicaltrials.gov/show/NCT00990002</url>
  </required_header>
  <id_info>
    <org_study_id>6002</org_study_id>
    <nct_id>NCT00990002</nct_id>
  </id_info>
  <brief_title>Investigation of A Children's Beverage Containing Different Probiotics</brief_title>
  <official_title>INVESTIGATION OF CHANGES IN LEUKOCYTE POPULATIONS AMONG CHILDREN RECEIVING A FOLLOW-ON COW'S MILK-BASED FORMULA CONTAINING DIFFERENT BIOACTIVE INGREDIENTS.</official_title>
  <sponsors>
    <lead_sponsor>
      <agency>Mead Johnson Nutrition</agency>
      <agency_class>Industry</agency_class>
    </lead_sponsor>
    <collaborator>
      <agency>Father Flanagan's Boys' Home</agency>
      <agency_class>Other</agency_class>
    </collaborator>
  </sponsors>
  <source>Mead Johnson Nutrition</source>
  <oversight_info>
    <has_dmc>No</has_dmc>
  </oversight_info>
  <brief_summary>
    <textblock>
      Investigating different bioactive ingredients in children's beverages and the effect on blood&#xD;
      serum markers.&#xD;
    </textblock>
  </brief_summary>
  <overall_status>Completed</overall_status>
  <start_date>September 2009</start_date>
  <completion_date type="Actual">May 2011</completion_date>
  <primary_completion_date type="Actual">May 2011</primary_completion_date>
  <study_type>Observational</study_type>
  <has_expanded_access>No</has_expanded_access>
  <study_design_info>
    <observational_model>Cohort</observational_model>
    <time_perspective>Prospective</time_perspective>
  </study_design_info>
  <number_of_groups>3</number_of_groups>
  <enrollment type="Anticipated">30</enrollment>
  <condition>Immunity</condition>
  <arm_group>
    <arm_group_label>Control</arm_group_label>
    <description>A children's milk-based beverage</description>
```

Relevant Fields

- Title
- ID
- Brief
- Description
- Gender
- Healthy
- Minage
- Maxage
- Condition
- Intervention
- Criteria

DATABASE



Database Preprocessing

1. Beautiful Soup and Regex were used to create a function that was able to parse through all the folders and extract these features: title, id, brief, description, gender, healthy, minage, maxage, condition, intervention, and criteria.
2. The features were combined as tuples and stored in a list resulting in > 16k items
3. The list was stored in a CSV file which is the basis for the final search engine index - it is uploaded as a dataframe and parsed as a list of tuples for ElasticSearch

DATABASE PARSER

```
def get_clinical_trials(data_files, topic_files):
    title = []
    id = []
    brief = []
    description = []
    gender = []
    healthy = []
    minage = []
    maxage = []
    condition = []
    intervention = []
    criteria = []
    inclusion = []
    exclusion = []
    topic_list = []


    for file in tqdm(data_files):
        with open(file, "r") as f:
            doc = BeautifulSoup(f, "html.parser")
            title.append(get_soup_text(doc, "title").replace('\n', ' ').replace('\r', ' '))
            id.append(get_soup_text(doc, "id").replace('\n', ' ').replace('\r', ' '))
            brief.append(get_soup_text(doc, "brief_summary").replace('\n', ' ').replace('\r', ' '))
            description.append(get_soup_text(doc, "description").replace('\n', ' ').replace('\r', ' '))
            gender.append(get_soup_text(doc, "gender").replace('\n', ' ').replace('\r', ' '))
            healthy.append(get_soup_text(doc, "healthy_volunteers").replace('\n', ' ').replace('\r', ' '))
            minage.append(get_soup_text(doc, "minimum_age").replace('\n', ' ').replace('\r', ' '))
            maxage.append(get_soup_text(doc, "maximum_age").replace('\n', ' ').replace('\r', ' '))
            condition.append(get_soup_text(doc, "condition_browse").replace('\n', ' ').replace('\r', ' '))
            intervention.append(get_soup_text(doc, "intervention").replace('\n', ' ').replace('\r', ' '))
            criteria.append(get_soup_text(doc, "criteria").replace('\n', ' ').replace('\r', ' '))
```

```
    for i in criteria:
        exclusion_match = re.search(r"Exclusion Criteria:(.*)", i)
        if exclusion_match:
            exclusion_criteria = exclusion_match.group(1).strip()
            exclusion.append(exclusion_criteria)

            inclusion_match = re.search(r"Inclusion Criteria:(.*?)Exclusion", i, re.DOTALL)
            if inclusion_match:
                inclusion_criteria = inclusion_match.group(1).strip()
                inclusion.append(inclusion_criteria)
            else:
                inclusion.append('')
        else:
            inclusion_match = re.search(r"Inclusion Criteria:(.*)", i, re.DOTALL)
            if inclusion_match:
                inclusion_criteria = inclusion_match.group(1).strip()
                inclusion.append(inclusion_criteria)


    tree = ET.parse(topic_files)
    topic_tags = tree.findall(".//topic")
    for tag in topic_tags:
        topic_list.append(tag.text.strip())



    return [(title, id, brief, description, gender, healthy, minage, maxage, condition, intervention, inclusion, exclusion)
            for title, id, brief, description, gender, healthy, minage, maxage, condition, intervention, inclusion, exclusion
            in zip(title, id, brief, description, gender, healthy, minage, maxage, condition, intervention, inclusion, exclusion)],
            topic_list
```

This original parser will not be included in the final submission

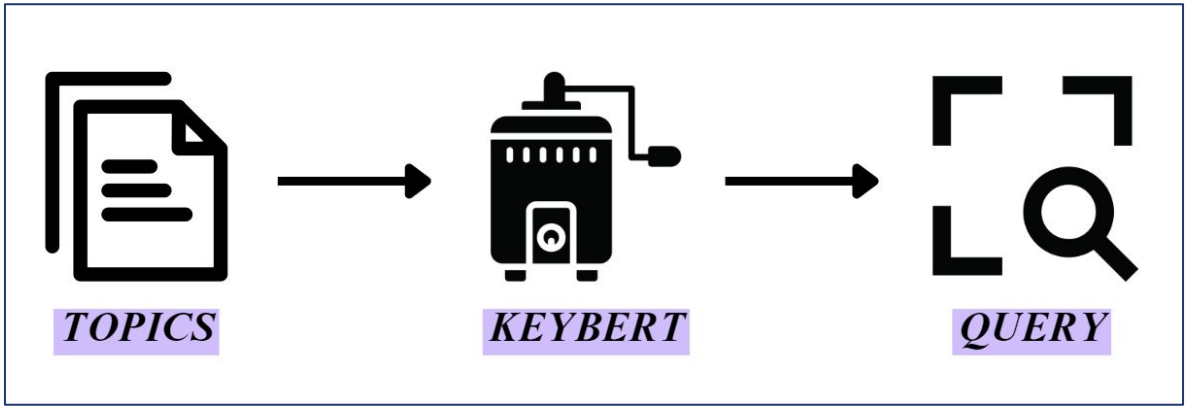
QUERIES

Features:

- Verbose
- Intended to emulate doctor notes
- Complex terminology
- Include treatment outline

```
<topics task="2021 TREC Clinical Trials">
<topic number="1">
Patient is a 45-year-old man with a history of anaplastic astrocytoma of the spine complicated by severe lower extremity weakness and urinary retention
s/p Foley catheter, high-dose steroids, hypertension, and chronic pain. The tumor is located in the T-L spine, unresectable anaplastic astrocytoma s/p
radiation. Complicated by progressive lower extremity weakness and urinary retention. Patient initially presented with RLE weakness where his right
knee gave out with difficulty walking and right anterior thigh numbness. MRI showed a spinal cord conus mass which was biopsied and found to be
anaplastic astrocytoma. Therapy included field radiation t10-l1 followed by 11 cycles of temozolomide 7 days on and 7 days off. This was followed by
CPT-11 Weekly x4 with Avastin Q2 weeks/ 2 weeks rest and repeat cycle.
</topic>
```

QUERIES



Query Preprocessing

1. The topic file is parsed with a function that transforms it into a list of topics
2. A random topic can be chosen using random.choice or a custom topic can be chosen to go through KeyBERT which will extract the most important keywords as specified
3. These keywords will act as the query for ElasticSearch

QUERY EXAMPLE

```
def query_maker(topics, top_n, relevant, non_relevant):
  ran_topic = topics[0]
  top_pos = int(topics.index(ran_topic)) + 1



  kw_model = KeyBERT()
  keywords = kw_model.extract_keywords(ran_topic, top_n= top_n, stop_words= 'english')
  query = [i[0] for i in keywords]


  age_list = [45, 48, 32, 44, 74, 55, 60, 57, 41, 22, 75, 34, 62, 70, 70, 79, 64, 78, 65,
         35, 57, 31, 39, 55, 42, 45, 53, 60, 24, 33, 37, 17, 42, 47, 15, 32, 20, 35,
         0.008, 60, 57, 19, 60, 14, 34, 30, 62, 41, 12, 0.42, 25, 34, 34, 57,
         22, 41, 41, 17, 15, 63, 45, 46, 54, 55, 25, 16, 54, 23, 67, 46, 34, 16, 0.008, 53, 55]

  query_age = age_list[top_pos - 1]

  gender_list = ['M', 'M', 'F', 'F', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'F', 'M', 'F', 'F', 'F',
                 'F', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'F', 'F', 'M', 'M', 'M', 'F', 'F', 'M',
                 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'M', 'M', 'F', 'F', 'M', 'M', 'M', 'M', 'M',
                 'F', 'M', 'F', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'F', 'M',
                 'M', 'F', 'F', 'M', 'F', 'F', 'F', 'F', 'F', 'M', 'M']

  gender = gender_list[top_pos - 1]

  top_rel = relevant[relevant['Topic_no'] == str(top_pos)]
  rel_doc = [i for i in top_rel["NCT_no"].astype('str')]

  non_rel = non_relevant[non_relevant['Topic_no'] == str(top_pos)]
  nonrel_doc = [i for i in non_rel["NCT_no"].astype('str')]

  return keywords, query, top_pos, query_age, gender, rel_doc, nonrel_doc
```

```
[64] bert, query, top_pos, age, gender, reldoc, nonrel_doc = query_maker(topics, 10, relevant, non_relevant)
     print(query)
     print(top_pos)
     print(age)
     print(gender)
     print(len(reldoc))
     print(len(nonrel_doc))

     ['astrocytoma', 'tumor', 'mri', 'spinal', 'spine', 'anaplastic', 'numbness', 'radiation', 'pain', 'extremity']
     1
     45
     M
     47
     360
```

Input Parameters

- Topics
- Number of keywords retrieved
- Relevance/Non-Relevance Dataframe

Output

- Keywords
- Topic number
- Age of patient query (mapped)
- Gender of patient query (mapped)
- Reldoc/Nonrel_doc

QUERY EXAMPLE: CONTROL

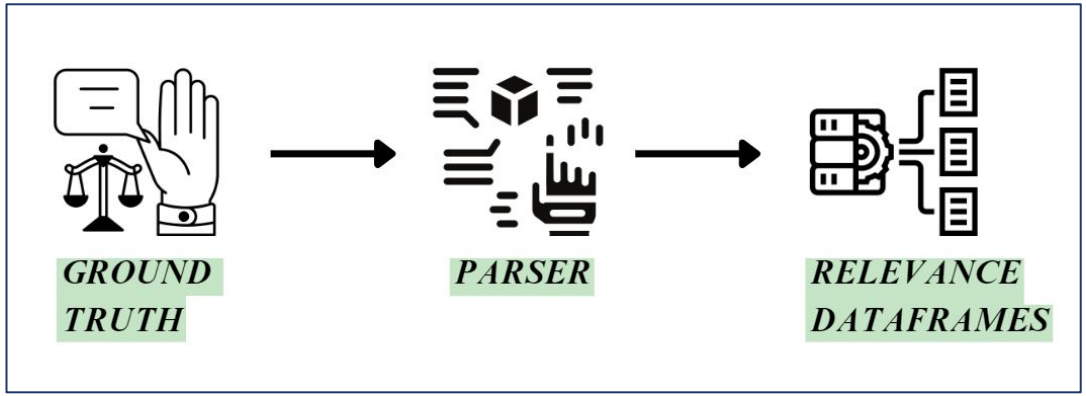
"Control" condition uses the entire query rather than keywords

- Offers comparison for performance of keyword search vs verbose query

Topics undergo following processes:

- Tokenization: sentences are split into words
- Stopword removal: removing words which appear commonly across the corpus. Standard english stopword dictionary is used from the nltk toolkit
- Lemmatization: each word is broken into its root meaning to identify similarities
    - E.g. "changing", "changes", "changed" are all lemmatized to "change"

RELEVANCE



35470	75	0	NCT01662427	2
35471	75	0	NCT01681641	2
35472	75	0	NCT01682837	2
35473	75	0	NCT01704742	0
35474	75	0	NCT01709513	1
35475	75	0	NCT01722578	0
35476	75	0	NCT01764815	2

Ground Truth Processing

1. The qrels2021 file was saved as a CSV and read into a function that extracts the 35k truths into a dataframe with columns for the various parameters
2. The parser function accepts a number as a parameter which retrieves a “relevant” dataframe based on the given relevance number, and also an additional “non-relevance” dataframe which is everything that is not the given relevance number

UTILISATION OF RELEVANCE

```python
def rel_df(qrelsdoc, relno):
    rel = None
    rel = pd.read_csv(qrelsdoc, header= None)
    relev = [i for i in rel[0]]
    columns = ["single"]

    final = None
    final = pd.DataFrame(relev, columns= columns)
    final[['Topic_no', '0', 'NCT_no', 'Relevance']] = final['single'].str.split(' ', n=3, expand=True)
    final = final.drop(columns=['single', '0'])
    relevant_rows = final[final['Relevance'] == str(relno)]
    non_relevant_rows = final[final['Relevance'] != str(relno)]


    return final, relevant_rows, non_relevant_rows
```

```
[23] reldf, relevant, non_relevant = rel_df('qrels2021.csv', 2)
```

```
[66] relevant
```

	Topic_no	NCT_no	Relevance
3	1	NCT00002814	2
10	1	NCT00003470	2
11	1	NCT00003471	2
12	1	NCT00003537	2
14	1	NCT00003621	2

The relevant/non-relevant dataframes are used to retrieve a list of topic IDs given the specific topic query chosen

```python
def query_maker(topics, top_n, relevant, non_relevant):
  ran_topic = topics[0]
  top_pos = int(topics.index(ran_topic)) + 1



  kw_model = KeyBERT()
  keywords = kw_model.extract_keywords(ran_topic, top_n= top_n, stop_words= 'english')
  query = [i[0] for i in keywords]


  age_list = [45, 48, 32, 44, 74, 55, 60, 57, 41, 22, 75, 34, 62, 70, 70, 79, 64, 78, 65,
         35, 57, 31, 39, 55, 42, 45, 53, 60, 24, 33, 37, 17, 42, 47, 15, 32, 20, 35,
         0.008, 60, 57, 19, 60, 14, 34, 30, 62, 41, 12, 0.42, 25, 34, 34, 57,
         22, 41, 41, 17, 15, 63, 45, 46, 54, 55, 25, 16, 54, 23, 67, 46, 34, 16, 0.008, 53, 55]

  query_age = age_list[top_pos - 1]

  gender_list = ['M', 'M', 'F', 'F', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'F', 'M', 'F', 'F', 'F',
                 'F', 'M', 'M', 'F', 'M', 'F', 'M', 'M', 'F', 'F', 'M', 'M', 'M', 'F', 'F', 'M',
                 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'M', 'M', 'F', 'F', 'M', 'M', 'M', 'M', 'M',
                 'F', 'M', 'F', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'M', 'M', 'F', 'M', 'F', 'M',
                 'M', 'F', 'F', 'M', 'F', 'F', 'F', 'F', 'F', 'M', 'M']

  gender = gender_list[top_pos - 1]

  top_rel = relevant[relevant['Topic_no'] == str(top_pos)]
  rel_doc = [i for i in top_rel["NCT_no"].astype('str')]

  non_rel = non_relevant[non_relevant['Topic_no'] == str(top_pos)]
  nonrel_doc = [i for i in non_rel["NCT_no"].astype('str')]

  return keywords, query, top_pos, query_age, gender, rel_doc, nonrel_doc
```

BACKGROUND: ELASTICSEARCH

- Based on Apache Lucene search engine library
- Indexing, searching and querying capabilities
- Stores data in JSON object format with chosen field mappings
- Indexing a document involves:
  - Breaking down into individual terms
  - Adding each term to an inverted index (maps each term to documents that contain this term)

- Queries are presented in a JSON query structure which specifies which fields to be searched
- Relevance scores are calculated using BM25 algorithm:
  - Uses term-frequency and inverse document-frequency
  - k: nonlinear term frequency saturation
  - b: normalisation of document length

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \left[\frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)} + \delta \right]$$

INDEXING DOCUMENT COLLECTION

- Index mapping created based on fields defined by document parsing function
- Indexed using ElasticSearch

```
request_body = {
    'settings': {
        'number_of_shards': 1,
        'number_of_replicas': 1,

    },
    'mappings': {
          'properties': {
              'title': {'type': 'text'},
              'id' : {'type': 'text'},
              'brief' : {'type': 'text'},
              'gender' : {'type': 'text'},
              'healthy' : {'type': 'text'},
              'minage' : {'type': 'float'},
              'maxage' : {'type': 'float'},
              'condition' : {'type': 'text'},
              'intervention' : {'type': 'text'},
              'inclusion' : {'type': 'text'},
              'exclusion' : {'type' : 'text'},
          }
    }
}

index_name = 'clinical_trials'
try:
  es.indices.get(index_name)
  print('index {} already exists'.format(index_name))
except:
  print('creating index {}'.format(index_name))
  es.indices.create(index_name, body=request_body)
```

EXPERIMENT TOPICS

Topic 1

- **Demographics**: 45 year-old male
- **History**: anaplastic astrocytoma of the spine complicated by severe lower extremity weakness and urinary retention. Complicated by progressive lower extremity weakness and urinary retention.
- **Presentation**: initially presented with RLE weakness where his right knee gave out with difficulty walking and right anterior thigh numbness.
- **Diagnosis**: anaplastic astrocytoma

Topic 49

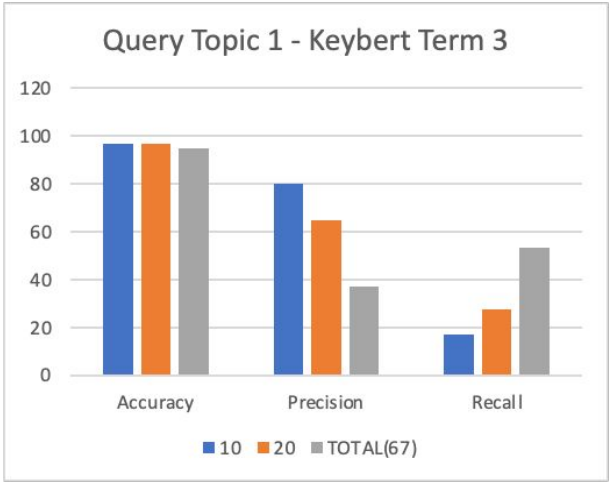
- **Demographics**: 12 year-old female
- **Presentation**: short stature, delayed in puberty and developmental delay
- **Diagnosis**: Turner syndrome
- **Additional factors:** obese, mentally retarded

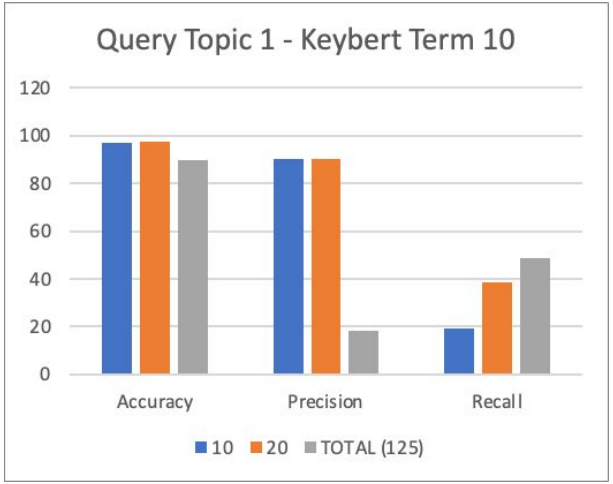
Topic 73

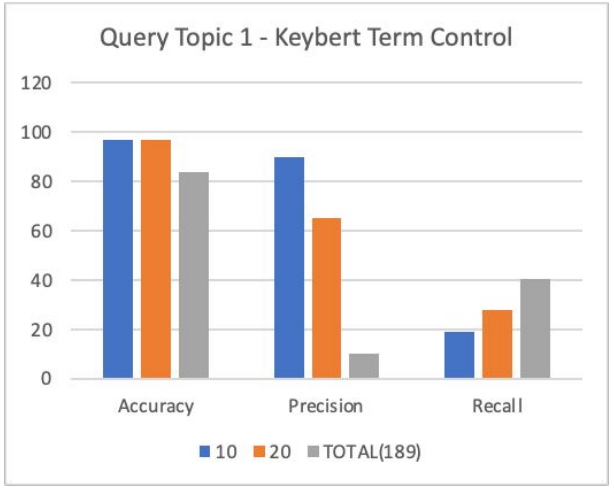
- **Demographics**: 3-day-old female
- **History**: born at 34w of gestation
- **Presentation**: yellow sclera and icteric body
- **Diagnosis**: neonatal jaundice

EXPERIMENT 1
KEYBERT TERM NUMBERS + NUMBER OF HITS

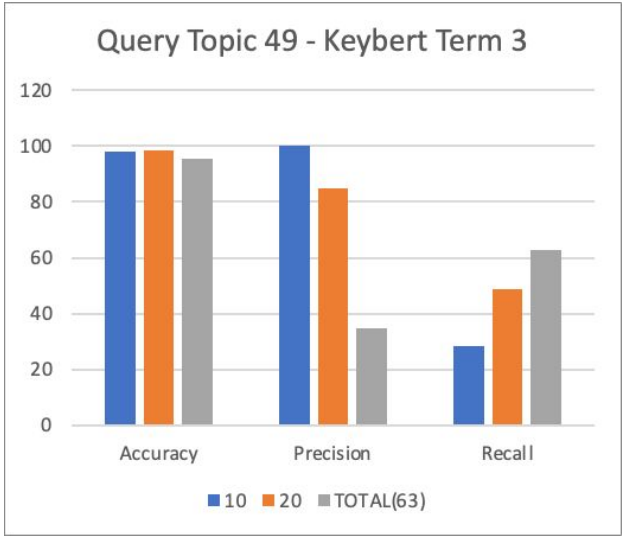
EXPERIMENT 1 RESULTS: TOPIC 1

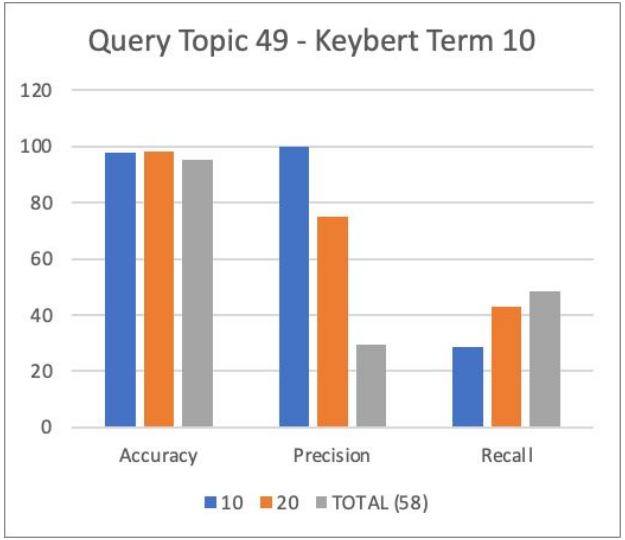


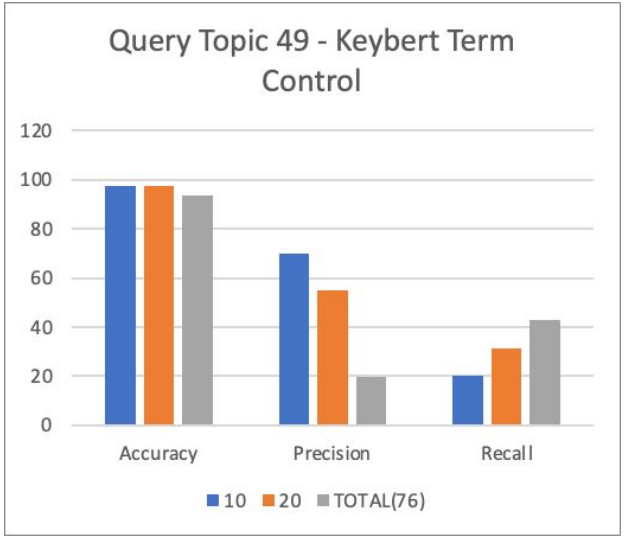




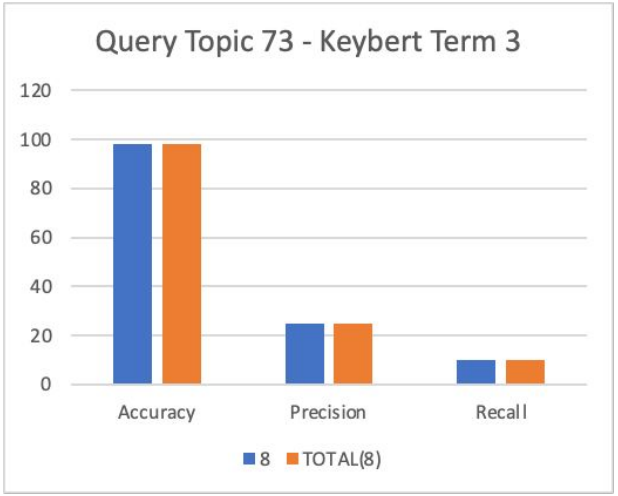
EXPERIMENT 1 RESULTS: TOPIC 49

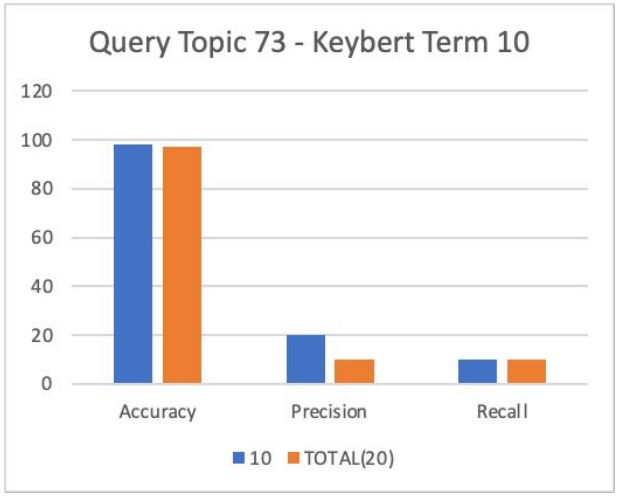


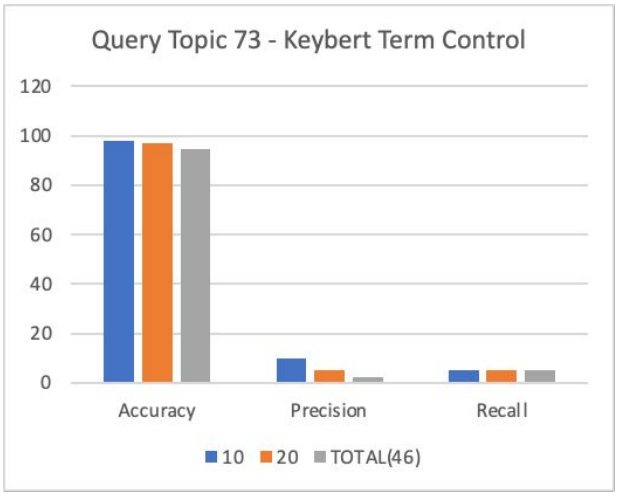




EXPERIMENT 1 RESULTS: TOPIC 73







EXPERIMENT 2
QUERY STRUCTURE

EXPERIMENT 2 - DIFFERENT QUERY STRUCTURES

Structure 1

One field only

- Title
- Inclusion

```
query_body = {
    "size": 10,
    'query':{
        'terms': {
            'title':  query
        }
    }
}
```

```
query_body = {
    "size": 10,
    'query':{
        'terms': {
            'inclusion':  query
        }
    }
}
```

Structure 2

Multi-match for multiple fields

- Title
- Inclusion
- Intervention

```
query_body = {
    "size": 10,
  "query": {
    "multi_match": {
      "query":  " ".join(query),
      "fields": ["title", "inclusion", "intervention"]
    }
  }
}
```

Structure 3

Multi-match for multiple fields

- Title
- Inclusion
- Intervention

Must match (boolean):

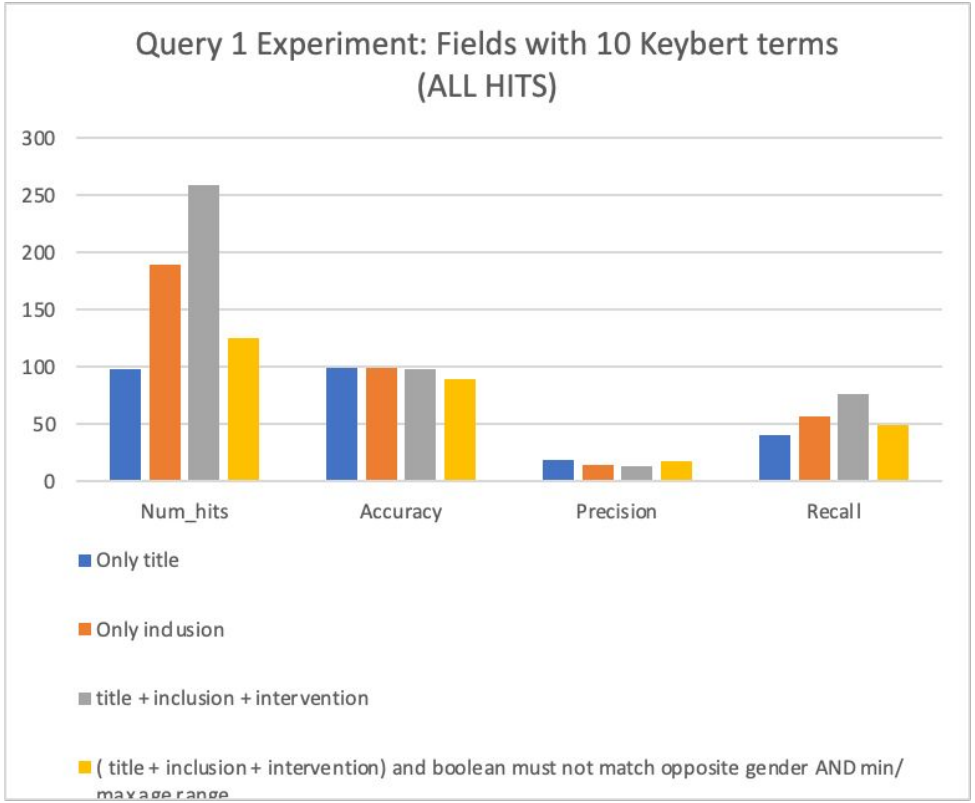
- Range:
- Minimum age ⩽ patient age
- Maximum age ⩾ patient age

Must not match(boolean)

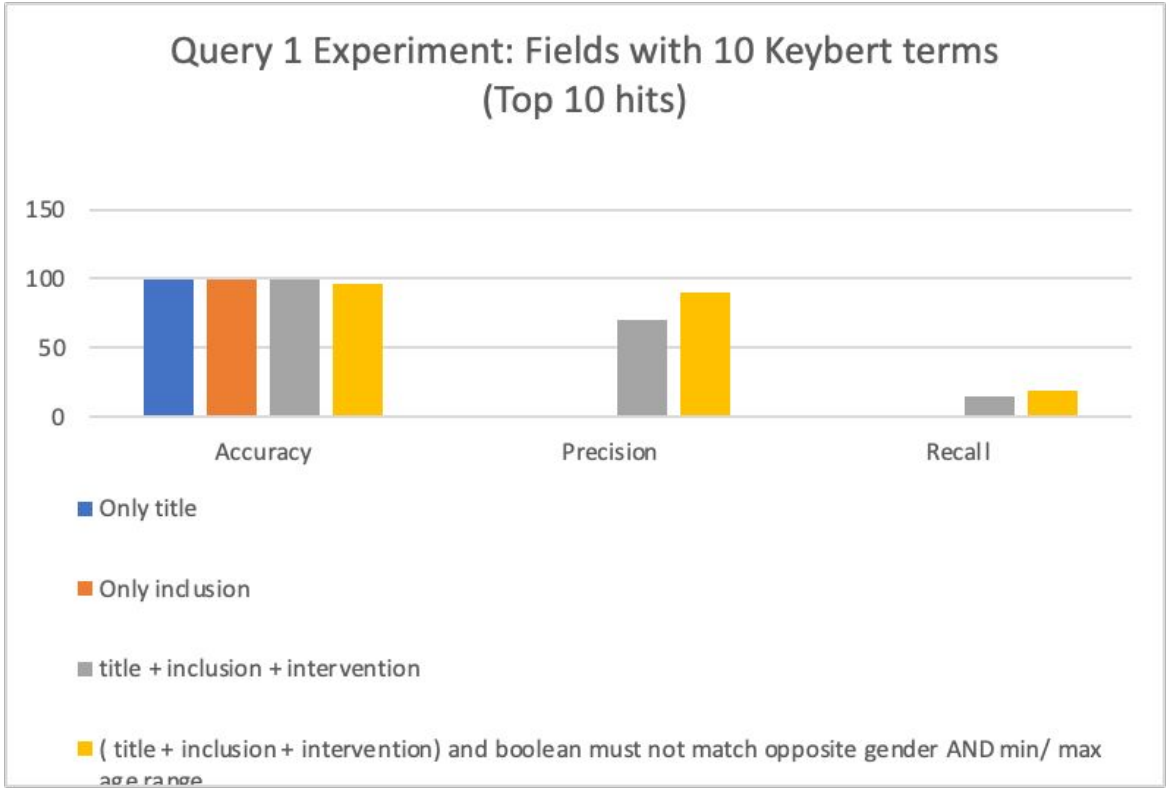
- Exclusion criteria
- Opposite gender of patient

```
query_body = {
  "size": 10,
  "query": {
    "bool": {
      "must": [
        {
          "range": {
            "minage": {
              "lte": age
            }
          }
        },
        {
          "range": {
            "maxage": {
              "gte": age
            }
          }
        },
        {
          "multi_match": {
            "query": " ".join(query),
            "fields": ["title", "inclusion", "intervention"]
          }
        }
      ],
      "must_not": [
        {
          "match": {
            "exclusion": " ".join(query)
          }
        },
        {
          "match": {
            "gender": oppo_gen
          }
        }
      ]
    }
  }
}
```

EXPERIMENT 2 RESULTS: ALL HITS



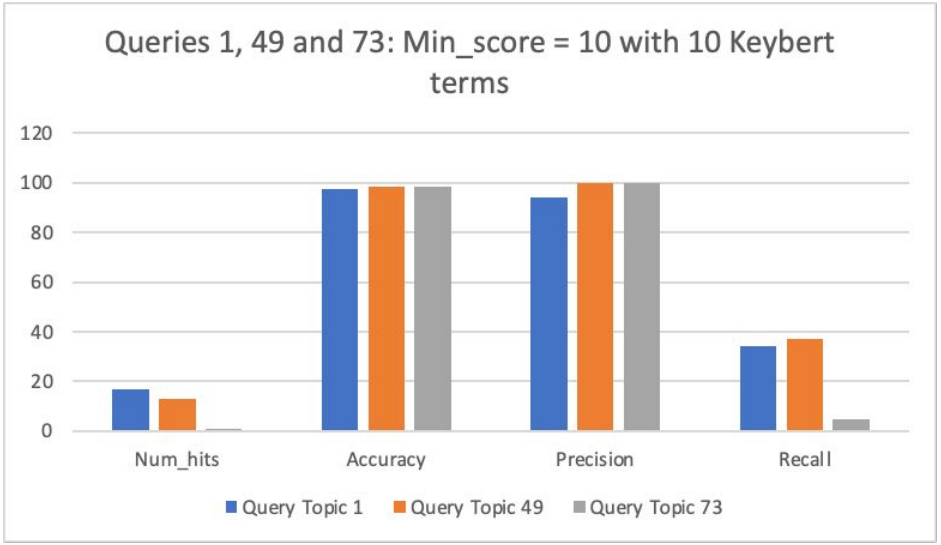
EXPERIMENT 2 RESULTS: TOP 10 HITS

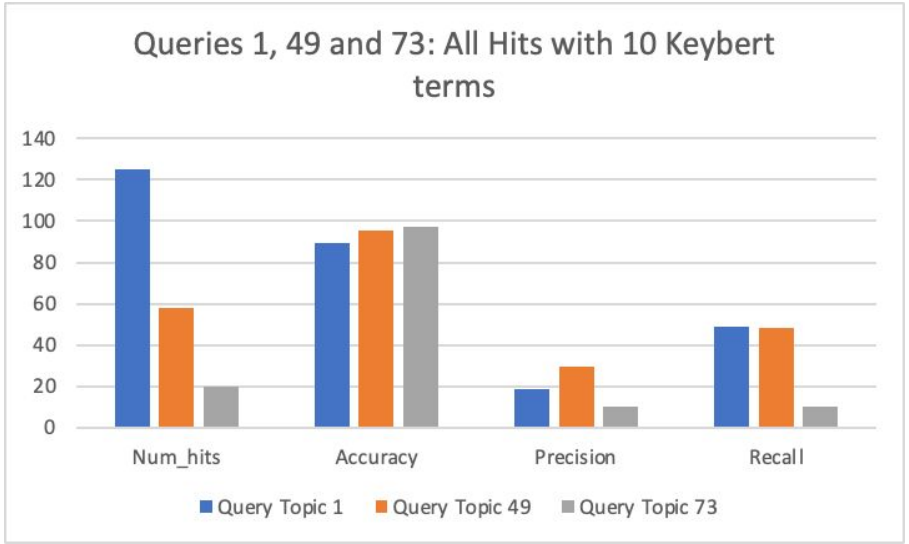


**Precision and Recall are 0 for “only title” and “only inclusion”

EXPERIMENT 3
MINIMUM SCORE THRESHOLD

EXPERIMENT 3 - RESULTS





ADDITIONAL EXPERIMENTS

Adjusting k and b parameters

- Negligible differences
  - Kept at default (b = 0.75, k = 1.2)

```
# Close the index
es.indices.close(index='clinical_trials')

# Define custom BM25 parameters
custom_bm25 = {
    "type": "BM25",
    "b": 0.75,
    "k1": 1.2
}

# Update the index settings
es.indices.put_settings(index='clinical_trials', body={
    "similarity": {
        "my_similarity": custom_bm25
    }
})

# Update the mapping of the index to use the custom similarity algorithm
es.indices.put_mapping(index='clinical_trials', body={
    "properties": {
        "my_field": {
            "type": "text",
            "similarity": "my_similarity"
        }
    }
})

# Reopen the index
es.indices.open(index='clinical_trials')
```

N-gram keyword extraction

[1, 2-gram, 3-gram]

- Negligible differences

```
query
['astrocytoma',
 'tumor',
 'mri',
 'spinal',
 'spine',
 'anaplastic',
 'numbness',
 'radiation',
 'pain',
 'extremity']
```

```
multi_kw_query
['astrocytoma spine',
 'anaplastic astrocytoma',
 'astrocytoma radiation',
 'astrocytoma therapy',
 'astrocytoma']
```

Boosting fields

- Title
- Inclusion
- Exclusion
- Harmed recall

```
{
  "multi_match": {
    "query": " ".join(query),
    "fields": ["title^10", "inclusion", "intervention"]
  }
```

```
"must_not": [
  {
    "match": {
      "exclusion^10": " ".join(query)
```

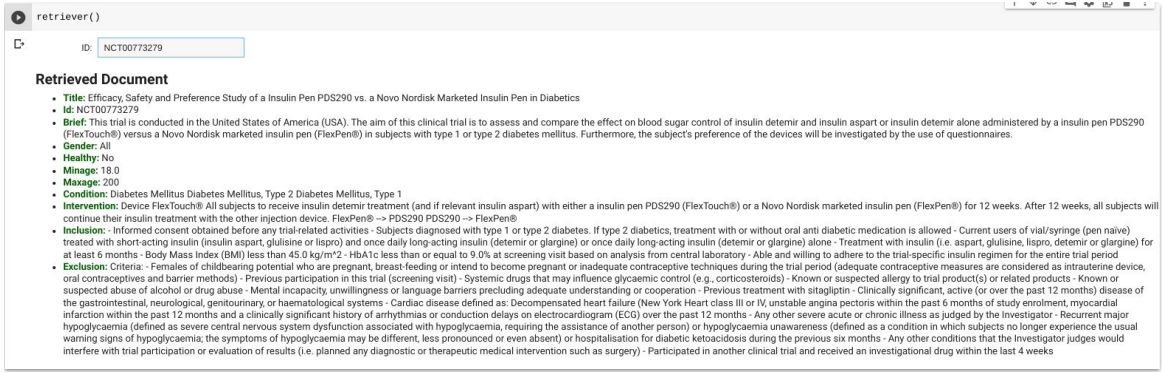
USER INTERFACE

Users can enter ID number from a retrieved document to access more information about the topic

- Useful for medical professionals who want to validate patient's suitability for trial

```
### RESULTS ####
Number of hits: 7

title: "Efficacy, Safety and Preference Study of a Insulin Pen PDS290 vs. a Novo Nordisk Marketed Insulin Pen in Diabetics", score: 17.770866,
 id: NCT00773279
```



LIMITATIONS

- Missing ground truth for many documents and resulting limited sample size
- Limited computational power for parsing entire document collection
- Potential loss of semantic meaning in search queries e.g. negatives “pregnant” vs “not pregnant”
- Inconsistency of document fields e.g. lack of inclusion/ exclusion criteria for some trials
- Need for “minimum threshold” of relevance to produce hits with high precision
- Need for domain experts to validate search results