Al ejecutar cualquier script es importante asegurarse de que se están ejecutando y descargando todo lo que necesitan en el mismo directorio donde están contenidos, ya que me dieron problemas al abrirlos todos a la vez con visual studio.

Para el ejercicio uno empecé escribiendo este código, usando un link de google drive creado por mi mismo para evitar tener que iniciar sesión en mi cuenta de la universidad.

```
import bm25s # To create indexes and search documents using BM25
import Stemmer  # For stemming terms
import json  # To load the JSON-formatted corpus
import gdown
import zipfile



link = "https://drive.google.com/uc?id=1n_L5EuLSWncIbxBqIG16ohbK9l09_sVq&export=download" #mismo archivo, pero lo subí a drive ya que el tuyo requería
archivo = "trec-covid-RI.zip"
gdown.download(link,archivo)
with zipfile.ZipFile(archivo, 'r') as zip_ref:
    zip_ref.extractall("trec-covid-RI")

with open("trec-covid-RI/corpus.jsonl","r",encoding="utf-8") as f:
    corpus_content = f.read()
    corpus_content = json.loads(corpus_content)
```

Lo cual me dio el siguiente error:

Exception has occurred: JSONDecodeError Extra data: line 2 column 1 (char 2121) File "C:\Users\drago\Desktop\Ejercicio-1.py", line 18, in <module> corpus_content = json.loads(corpus_content) json.decoder.JSONDecodeError: Extra data: line 2 column 1 (char 2121)

Ya que el formato del archivo es jsonlines y no json, por lo que le pregunté a chat GPT como arreglarlo,pasándole este error como prompt y me dijo que cambiase el código a esto y funcionó:

```python
import bm25s # To create indexes and search documents using BM25
#import Stemmer  # For stemming terms
import json  # To load the JSON-formatted corpus
import gdown
import zipfile



link = "https://drive.google.com/uc?id=1n_L5EuLSWncIbxBqIG16ohbK9l09_sVq&export=download" #mismo archivo, pero lo subí a drive ya que el tuyo requería iniciar sesión.
archivo = "trec-covid-RI.zip"
gdown.download(link,archivo)
with zipfile.ZipFile(archivo, 'r') as zip_ref:
    zip_ref.extractall("trec-covid-RI")

corpus_content = []
with open("trec-covid-RI/corpus.jsonl","r",encoding="utf-8") as f:
    for line in f:
        corpus_content.append(json.loads(line))
```

Para terminar el ejercicio añadí una comprobación para no tener que volver a descargar el archivo si este no existe y copié y pegué el código de los colab de clase para indexar.

En el ejercicio 2 parto originalmente del código del ejercicio anterior, en un primer momento escribí el método

```
def indexar(bm25_flavour,idf_flavour,palabras_vacias,stemming,nombre):
    stemmer = Stemmer.Stemmer("english") if stemming else None
    if(palabras_vacias):
        if(stemming):
            corpus_tokenized = bm25s.tokenize(corpus_plaintext, stopwords="en", stemmer=stemmer, show_progress=True)
        else:
            corpus_tokenized = bm25s.tokenize(corpus_plaintext, stopwords="en", show_progress=True)
    else:
        if(stemming):
            corpus_tokenized = bm25s.tokenize(corpus_plaintext, stemmer=stemmer, show_progress=True)
        else:
            corpus_tokenized = bm25s.tokenize(corpus_plaintext,show_progress=True)
    retriever = bm25s.BM25(corpus=corpus_verbatim, method=bm25_flavour, idf_method=idf_flavour)
    retriever.index(corpus_tokenized, show_progress=True)
    retriever.save(nombre, corpus=corpus_verbatim)
```

Que pretendía meter dentro de un par de bucles, pero como no estaba seguro de que hacer con los nombres le pedí a chat gpt que me ayudase con el prompt:

def indexar(bm25_flavour,idf_flavour,palabras_vacias,stemming,nombre): if(palabras_vacias): if(stemming): corpus_tokenized = bm25s.tokenize(corpus_plaintext, stopwords="en", stemmer=stemmer, show_progress=True) else: corpus_tokenized = bm25s.tokenize(corpus_plaintext, stopwords="en", show_progress=True) else: if(stemming): corpus_tokenized = bm25s.tokenize(corpus_plaintext, stemmer=stemmer, show_progress=True) else: corpus_tokenized = bm25s.tokenize(corpus_plaintext,show_progress=True) retriever = bm25s.BM25(corpus=corpus_verbatim, method=bm25_flavour, idf_method=idf_flavour) retriever.index(corpus_tokenized, show_progress=True) retriever.save(nombre, corpus=corpus_verbatim) Como se vio en el segundo notebook, bm25s admite cinco variantes de BM25 e IDF: robertson, atire, bm25l, bm25+ y lucene. Además, el texto puede preprocesarse eliminando (o no) palabras vacías y aplicando (o no) stemming. En este ejercicio, se debe desarrollar un código que genere “rondas” de resultados con 100 documentos (solo los identificadores) para todas las consultas de la colección. Dichas rondas deben ejecutarse considerando todas las combinaciones posibles de parámetros. Además, los resultados de cada ronda deben guardarse en el disco con nombres descriptivos (p.ej., robertson-stopwords-stemming, bm25plus-NON-stopwords-stemming, lucene-NON-stopwords-NON-stemming, etc.) y subirse al campus virtual. Cada archivo de resultados se evaluará en términos de precisión, exhaustividad y F1, utilizando los juicios de relevancia de la colección. La documentación incluirá una tabla mostrando los resultados, tanto micro como macro-promediados, destacando en negrita los valores de F1 macro-promediados. Con base en dichos resultados, se seleccionará la parametrización que obtenga el mejor F1 macro-promediado, la cual se indicará en la documentación y se utilizará en el tercer ejercicio. Puedes ajustar este código mejor al enunciado

Y tras ajustar el código que me dio me quedó esto

```python
def indexar():
    for bm25_flavor in bm25_flavours:
            for stopwords in [True, False]:
                for stem in [True, False]:
                    if stopwords:
                        if stem:
                            corpus_tokenized = bm25s.tokenize(corpus_plaintext, stopwords="en", stemmer=stemmer, show_pr
                        else:
                            corpus_tokenized = bm25s.tokenize(corpus_plaintext, stopwords="en", show_progress=True)
                    else:
                        if stem:
                            corpus_tokenized = bm25s.tokenize(corpus_plaintext, stemmer=stemmer, show_progress=True)
                        else:
                            corpus_tokenized = bm25s.tokenize(corpus_plaintext, show_progress=True)
                    stopwords_str = "stopwords" if stopwords else "NON-stopwords"
                    stemming_str = "stemming" if stem else "NON-stemming"
                    file_name = f"{bm25_flavor}-{stopwords_str}-{stemming_str}"
                    retriever = bm25s.BM25(corpus=corpus_verbatim, method=bm25_flavor, idf_method=bm25_flavor)
                    retriever.index(corpus_tokenized, show_progress=True)
                    retriever.save(file_name, corpus=corpus_verbatim)
```

(Asumí que method e idf_method son siempre lo mismo ya que en las instrucciones se menciona que solo hay 20 combinaciones posibles)

Cogí el método submit_queries_and_get_run, modificándolo un poco para hacer la tokenización dependiendo del retriever que se le pase (ajustando stem y stopwords, como en el caso anterior), además de pasar a llamar este metodo tras crear un retriever en el metodo indexar.

Para obtener el array relevance_judgements tuve que leer el archivo qrel.tsv, como no sabía como leer este tipo de archivos se lo pregunté a chat gpt con el prompt

with open("lisa-relevance-judgements.json", "r", encoding="utf-8") as f: relevance_judgements = json.load(f) # The format of the relevance judgments need to be slightly transformed relevance_judgements_reformat = dict() for entry in relevance_judgements: query_id = entry["query_id"] rel_docs = entry["rel_docs"].split(", ") relevance_judgements_reformat[query_id] = rel_docs como puedo hacer esto para un archivo qrels.tsv de formato query-id corpus-id score 1 005b2j4b 2

Lo que, tras unos ajuste llegué al código

```
with open(qrels_file, "r", encoding="utf-8") as f:
    reader = csv.reader(f, delimiter="\t")
    next(reader)
    for row in reader:
        query_id, corpus_id, score = row
        if query_id not in relevance_judgements_reformat:
            relevance_judgements_reformat[query_id] = []
        relevance_judgements_reformat[query_id].append(corpus_id)
```

Y finalmente para acabar el ejercició copié y pegué el método compute_precision_recall_f1 de los apuntes y lo llamo al final de cada ejecución de submit_queries_and_get_run, haciendo que en lugar de imprimar añada los valores a un archivo medidas.json, para lo que en un primer lugar, los guardaba en un archivo .txt sin más y luego le pedí a chat gpt que lo pasase a json con el prompt

*def compute_precision_recall_f1(run, relevance_judgements,nombre): with open(nombre+"/medidas.txt", "w", encoding="utf-8") as f: # Initialize lists to hold precision, recall, and f1 scores for each query precision_values = [] recall_values = [] f1_values = [] # Initialize global counts for micro-averaging global_retrieved = 0 global_relevant = 0 global_retrieved_and_relevant = 0 # Compute precision, recall, and F1 score for each query for query_id in run.keys(): retrieved_results = run[query_id] relevant_results = relevance_judgements[query_id] relevant_and_retrieved = set(retrieved_results) & set(relevant_results) # Update global counts global_retrieved += len(retrieved_results) global_relevant += len(relevant_results) global_retrieved_and_relevant += len(relevant_and_retrieved) # Compute precision and recall precision = len(relevant_and_retrieved) / len(retrieved_results) if len(retrieved_results) > 0 else 0 recall = len(relevant_and_retrieved) / len(relevant_results) if len(relevant_results) > 0 else 0 # Compute F1 score if both precision and recall are non-zero if (precision + recall) > 0: f1 = 2 * (precision * recall) / (precision + recall) f1_values.append(f1) # Append precision and recall for the current query precision_values.append(precision) recall_values.append(recall) # Compute macro-averages macro_average_precision = sum(precision_values) / len(precision_values) if precision_values else 0 macro_average_recall = sum(recall_values) / len(recall_values) if recall_values else 0 macro_average_f1 = sum(f1_values) / len(f1_values) if f1_values else 0 # Print macro-averaged scores f.write(f"Macro-averaged Precision: {round(macro_average_precision, 3)}\n") f.write(f"Macro-averaged Recall: {round(macro_average_recall, 3)}\n") f.write(f"Macro-averaged F1: {round(macro_average_f1, 3)}\n\n") # Compute micro-averages micro_average_precision = global_retrieved_and_relevant / global_retrieved if global_retrieved > 0 else 0 micro_average_recall = global_retrieved_and_relevant / global_relevant if global_relevant > 0 else 0 micro_average_f1 = (2 **

*(micro_average_precision * micro_average_recall) / (micro_average_precision + micro_average_recall)) if (micro_average_precision + micro_average_recall) > 0 else 0 # Print micro-averaged scores f.write(f"Micro-averaged Precision: {round(micro_average_precision, 3)}\n") f.write(f"Micro-averaged Recall: {round(micro_average_recall, 3)}\n") f.write(f"Micro-averaged F1: {round(micro_average_f1, 3)}\n\n") puedes hacer que se guarde en un archivo .json,*

Lo que me devolvió el código que tengo puesto.

Después de esto le pedí a chat gpt que me crease métodos para comprobar que existiesen las colecciones creadas en la misma carpeta, con el prompt

(codigo entero) indexar() puedes crearme una lista con todos los posibles nombres generados y hacer una condicion que compruebe que no exista ninguno

Y subí todas las carpetas creadas anteriormente a drive, y en caso de no existir una carpeta "colecciones" en el directorio del script las descargo y descomprimo a una carpeta colecciones dentro del directorio del script.

```
def descargarTodo():
    for name in possible_names:
        directory_path = os.path.join("colecciones", name)
        if not os.path.isdir(directory_path):
            descargar()
    #print(name)
    retriever = bm25s.BM25.load(directory_path)
    stopwords = "NON-stopwords" not in name
    stem = "NON-stemming" not in name
    submit_queries_and_get_run2(all_queries,stemmer,retriever,stopwords,stem,name)
```

Escribí este código para descargar los archivos creados,luego me daba error en submit_queries_and_get_run por lo que cambié el corpus de retriever.retrieve por corpus_vertabim y lo moví a un nuevo método submit_queries_and_get_run2, y ya funcionaba todo correctamente. Ahora, si se quiere volver a indexar habría que llamar a indexar() en lugar de a descargarTodo()

Lo que nos deja la tabla (Siendo non stop words las que no eliminan las palabras vacias):

Medida	Macro-averaged	Macro-averaged Recall	Macro-averaged F1	Micro-averaged Precision	Micro-averaged Recall	Micro-averaged F1

	Precision					
lucene-NON-stopwords-NON-stemming	0.718	0.061	**0.111**	0.718	0.054	0.101
lucene-stopwords-NON-stemming	0.718	0.061	**0.111**	0.718	0.054	0.101
lucene-NON-stopwords-stemming	0.756	0.063	**0.116**	0.756	0.057	0.106
lucene-stopwords-stemming	0.756	0.063	**0.116**	0.756	0.057	0.106
robertson-NON-stopwords-NON-stemming	0.713	0.06	**0.11**	0.713	0.054	0.1
robertson-stopwords-NON-stemming	0.713	0.06	**0.11**	0.713	0.054	0.1
robertson-NON-stopwords-stemming	0.752	0.063	**0.116**	0.752	0.057	0.105
robertson-stopwords-stemming	0.752	0.063	**0.116**	0.752	0.057	0.105
atire-NON-stopwords-NON-stemming	0.718	0.061	**0.111**	0.718	0.054	0.101
atire-stopwords-NON-stemming	0.718	0.061	**0.111**	0.718	0.054	0.101
atire-NON-stopwords--stemming	0.756	0.063	**0.116**	0.756	0.057	0.106
atire-stopwords-stemming	0.756	0.063	**0.116**	0.756	0.057	0.106

bm25l-NON-stopwords-NON-stemming	0.714	0.06	**0.111**	0.714	0.054	0.1
bm25l-stopwords-NON-stemming	0.714	0.06	**0.111**	0.714	0.054	0.1
bm25l-NON-stopwords-stemming	0.75	0.063	**0.116**	0.75	0.057	0.105
bm25l-stopwords-stemming	0.75	0.063	**0.116**	0.75	0.057	0.105
bm25+-NON-stopwords-NON-stemming	0.718	0.061	**0.111**	0.718	0.054	0.101
bm25+-stopwords-NON-stemming	0.718	0.061	**0.111**	0.718	0.054	0.101
bm25+-NON-stopwords-stemming	0.756	0.063	**0.116**	0.756	0.057	0.106
bm25+-stopwords-stemming	0.756	0.063	**0.116**	0.756	0.057	0.106

Como conclusión de estos resultados podemos ver que la eliminación de las palabras vacías no tiene ningún efecto (probablemente por que son artículos científicos, donde se utiliza un vocabulario concreto, además de que estoy usando idf en todas las ejecuciones) y que el stemming tiene un efecto positivo en todos los casos. En cuanto a la mejor parametrización f1 macro promedio empatan todas al hacer stemming.

Para el ejercicio 3 empezé partiendo del código del ejercicio anterior y pidiéndole a chat gpt que me pasase el código de java para calcular el rootlikelihood a python pasándole como prompt el código entero de la clase y pidiéndole al final que lo pasase a python.

Después de esto modifiqué el método submit_queries_and_get_run phice que el método indexar solo usase bm25+ con stemming y stopwords(elegido al azar ya que todos los flavours con stemming tenían la misma precisión), para calcular term frequency totales copié y pegué el método que nos diste en el campus, y para

calcularlas dentro de los resultados primero cree run de esta manera

```
run = {
    "ids": [],
    "cuerpos": []
}

for doc in returned_documents:
    doc_id = str(doc["id"])
    cuerpo = doc["text"]

    run["ids"].append(doc_id)
    run["cuerpos"].append(cuerpo)
```

y luego le pedí a chat gpt que me ayudase a calcular la term frequency con este prompt {'ids': ['wuegn0jg', 'ne5r4d4b', '75773gwg', 'e3wjo0yk', 'kqqantwg'], 'cuerpos': ['infections with bat-origin coronaviruses have caused severe illness in humans by ‘hos...eat to both agriculture and public health.', 'severe acute respiratory syndrome coronavirus (sars-cov) and middle east respiratory ...a syndrome coronavirus (sads-cov) to pigs.', 'the ongoing pandemic of coronavirus disease 2019 (covid-19), caused by infection with...he control and prevention of the pandemic.', 'for decades, french guinea fowl have been affected by fulminating enteritis of unclea... caused by coronaviruses of animal origin.', 'we showed that severe acute respiratory syndrome coronavirus 2 is probably a novel re...ncestral viruses have not been identified.']} run es asi, quiero que el metodo cuente el total de veces que se repite cada palabra en todos los cuerpos def compute_term_frequencies_from_corpus_tokenized(corpus_tokenized): tmp = dict() for document in corpus_tokenized[0]: freqs = dict(Counter(document)) for token, freq in freqs.items(): try: tmp[token] += freq except: tmp[token] = freq inverted_vocab = {corpus_tokenized[1][key]: key for key in corpus_tokenized[1].keys()} total_freqs = dict() for key, freq in dict(tmp).items(): term = inverted_vocab[key] total_freqs[term] = freq,

Lo que me dio este método

```python
def calculate_term_frequencies_from_run(run):

    import re
    tmp = dict()


    for cuerpo in run["cuerpos"]:

        tokens = re.findall(r'\b\w+\b', cuerpo.lower())
        freqs = dict(Counter(tokens))


        for token, freq in freqs.items():
            try:
                tmp[token] += freq
            except KeyError:
                tmp[token] = freq

    return tmp
```

Con el que consigo la segunda colección term_frequencies_run

Para conseguir la tercera colección term_frequencies_resto creé una copia del term_frequencies_totales,con todos los documentos y le resté los obtenidos previamente por la consulta, llamé al método compare_frequencies,hecho por chat gpt previamente para calcular los valores llr, pasándole term_frequencies_run como primera colección y term_frequencies_resto como segunda.

Luego hice un loop para conseguir los m primeros términos con mayor llr comprobando que no están en el query inicial para finalmente añadir estos elementos al query y volver a tokemizar y llamar a bm25,esta vez con 100 documentos para poder comparar con el ejercicio anterior, no estaba seguro de como hacerlo así que se lo pregunté a chat gpt con el prompt:

Esos m términos se añaden a la consulta inicial, y se vuelve a enviar al índice BM25S para obtener la lista definitiva de resultados. una vez que tengo una lista resultados de estructura 'clinic': 63144 como puedo hacer esto.

Y para acabar calculo los valores F1 como siempre.

(En este ejercicio uso queries parafraseadas y en español, explico en el 5 de donde las saco y por qué)

Para empezar el ejercicio 4 borré los inputs para n y m, creé dos loops, el primero para n en indexar y el segundo para m dentro de submit_queries_and_get_run e hice todo como en el ejercicio anterior, menos al final que se vuelve a crear el diccionario run con los documentos devueltos por las nuevas querys,además cambié el número de documentos por 100 para tener medidas más representativas y se calculan sus valores de precision, recall y f1,guardándolos en un documento de texto distinto para cada valor de n y m distinto, lo que dio los siguientes resultados:

Medida	Macro-averaged F1
m = 3 n = 1	0.109
m = 3 n = 2	0.11
m = 3 n =3	0.111
m = 3 n = 4	0.114
m = 3 n = 5	0.115
m = 4 n = 1	0.106
m = 4 n = 2	0.11
m = 4 n = 3	0.111
m = 4 n = 4	0.112
m = 4 n = 5	0.113
m = 5 n = 1	0.101
m = 5 n = 2	0.106
m = 5 n = 3	0.11

m = 5 n = 4	0.11
m = 5 n = 5	0.112

Como conclusión de estos resultados podemos deducir que para valores mayores de m o bien se mantienen las medidas iguales o baja la medida F1.

Para mayores valores de n esta sube en todos los casos, por lo que la mejor opción sería m=3 y n = 5.

En comparación con la mejor medida tomada anterior mente (cualquier flavour con stemming) podemos ver que son prácticamente iguales pero un poco peor,teniendo aquí una mejor medida de 0.115 en comparación a 0.116 del ejercicio dos, pero al ser un cambio mínimo, menor del 5% lo podemos descartar y podemos decir que para esta colección tendrían el mismo rendimiento. Si echamos un vistazo rápido a los queries individuales podemos ver que para algunos es mucho mejor este enfoque, llegando a subir hasta 0,5 mientras que en otros pasa al revés, llegando a bajar hasta 0,5.

En el ejercicio 5 empecé partiendo del código anterior, borré todo el código extra que no necesitaba y copié y pegué el código de los apuntes, estructurándolo para solo se cree tanto el embedding como la colección si esta no se encuentra en la misma carpeta que el script,una vez creada la colección como tarda mucho en crearse la subí a drive para descargarla directamente, de manera similar al ejercicio dos, si se desea volver a crear en lugar de descargarla habrá que llamar a este método dentro de if(not existe), imp:

```
def crear():
    chromadb_documents = []
    chromadb_doc_ids = []

    for document in corpus_content:
    #print(document)
        doc_id = str(document["_id"])
        title = document["title"].lower()
        content = document["text"].lower()

        chromadb_doc_ids.append(doc_id)
        chromadb_documents.append(f"{title} {content}")

    chromadb_embeddings = model.encode(chromadb_documents, batch_size=100, show_progress_bar=True)#, device='cuda')


    document_batches = get_batches(chromadb_documents)
    ids_batches = get_batches(chromadb_doc_ids)
    embedding_batches = get_batches(chromadb_embeddings)
    for i in range(len(document_batches)):
       documents = document_batches[i]
       doc_ids = ids_batches[i]
       embeddings = embedding_batches[i]
       collection.add(documents=documents,ids=doc_ids,embeddings=embeddings)
```

Y si se quiere descargar, para que sea más rápido hay que llamar a este:

```
def descargar():
    link = "https://drive.google.com/uc?id=1Me1JIbrMOJtHLj7e0u_yjfaQnZM0BwvN&export=download"
    archivo = "chromadb-storage.zip"
    gdown.download(link,archivo)
    with zipfile.ZipFile(archivo, 'r') as zip_ref:
        zip_ref.extractall("chromadb-storage")
```

Después de descargarla o crearla se para la ejecución y se tiene que volver a llamar para calcular las medidas f,por que si no no la detecta como creada.

Para el número de documentos usé 100 otra vez para tener una comparación justa con el resultado del último ejercicio, y creo los runs como en los apuntes:

```python
def submit_queries_and_get_run(queries, collection, max_results, nombre):
    run = {}
    for query in queries:
        query_id = query["_id"]
        query_string = query["metadata"]["query"].lower()

        results = collection.query(
            query_texts=[query_string],
            n_results=max_results
        )
        run[query_id] = results['ids'][0]


    compute_precision_recall_f1(run, relevance_judgements_reformat, nombre)
```

Además, usando claude cree dos nuevos documentos para los queries, el primero de ellos pasándole como prompt el archivo entero y pidiéndole que le tradujese al español sin cambiar el formato y para el segundo “basándote en el archivo original en inglés puedes parafrasearlo para que use un lenguaje más común (en ingles)”. Guardando los resultados en los archivos queries_paraphrased.jsonl y queries_spanish.jsonl dentro de la carpeta trec-covid-RI.

Además, volví hacia atrás para calcular los resultados f1 macro promedios para el ejercicio 4 con las queries parafraseadas y en español.

También actualicé el link de descarga de la colección en todos los archivos para que contenga las nuevas queries.

Así que, para el ejercicio 4 tenemos las siguientes tablas:

Queries por defecto

Medida	Macro-averaged F1
m = 3 n = 1	0.109
m = 3 n = 2	0.11
m = 3 n =3	0.111
m = 3 n = 4	0.114
m = 3 n = 5	0.115
m = 4 n = 1	0.106
m = 4 n = 2	0.11

m = 4 n = 3	0.111
m = 4 n = 4	0.112
m = 4 n = 5	0.113
m = 5 n = 1	0.101
m = 5 n = 2	0.106
m = 5 n = 3	0.11
m = 5 n = 4	0.11
m = 5 n = 5	0.112

Queries en español (cambiando stemmer a spanish)

Medida	Macro-averaged F1
m = 3 n = 1	0.014
m = 3 n = 2	0.014
m = 3 n =3	0.014
m = 3 n = 4	0.014
m = 3 n = 5	0.014
m = 4 n = 1	0.014
m = 4 n = 2	0.015
m = 4 n = 3	0.015
m = 4 n = 4	0.015
m = 4 n = 5	0.015
m = 5 n = 1	0.015
m = 5 n = 2	0.014
m = 5 n = 3	0.015

m = 5 n = 4	0.015
m = 5 n = 5	0.015

Queries parafraseadas

Medida	Macro-averaged F1
m = 3 n = 1	0.11
m = 3 n = 2	0.11
m = 3 n =3	0.114
m = 3 n = 4	0.116
m = 3 n = 5	0.115
m = 4 n = 1	0.107
m = 4 n = 2	0.11
m = 4 n = 3	0.112
m = 4 n = 4	0.113
m = 4 n = 5	0.115
m = 5 n = 1	0.103
m = 5 n = 2	0.108
m = 5 n = 3	0.108
m = 5 n = 4	0.111
m = 5 n = 5	0.114

Mientras que para el ejercio 5 tenemos las siguientes medidas:

Queries	Macro-averaged F1
Por defecto	0.023
Español	0.019
Parafraseadas	0.023

Que, si las comparamos con la mejor medida medida macro promedio F1 del ejercicio anterior, que son estas:

Queries	Macro-averaged F1
Por defecto	0.115
Español	0.015
Parafraseadas	0.116

Podemos ver que para las queries en inglés escritas con lenguaje técnico y para las parafraseadas,también en inglés es mucho peor, teniendo una medida macro promedio F1 80% peor en comparación con la del ejercicio anterior.

Mientras que para las queries traducidas al español tiene una medida macro promedio F1 26,67% mejor, aunque sigue siendo un resultado bastante malo, es mucho mejor que la estrategia de bm25+ con stemming y expansión de consultas.

Por lo que, como conclusión, podemos decir que es mejor usar la búsqueda semántica si tenemos queries en español, mientras que en el caso de tener queries en inglés podríamos alternar entre cualquier flavour de bm25 con stemming y con o sin expansión de consultas y con o sin palabras vacías, con m=3 y n= 5, ya que tienen resultados prácticamente iguales.