Ejercicios

November 30, 2024

1 Ejercicio 1

En este primer ejercicio procedemos a indexar la colección de documentos `train.docs`, dicha colección será tokenizada y se creará un índice BM25.

Primero empezamos importando las dependencias necesarias:

[ ]:
```
import bm25s
import Stemmer
```

Estas librerías nos permiten: - bm25s: Crear índices y buscar documentos usando BM25 - Stemmer: Realizar stemming de los términos

Una vez importadas dichas librerías necesitamos carga el corpus para ello se define la siguiente función:

[ ]:
```
def prepareCorpus():

    # Cargamos el archivo train.docs, y lo separamos por el salto de línea
    with open("../archivos/train.docs","r",encoding="utf-8") as f:
        corpus_content = f.read()
        corpus_content = corpus_content.split("\n")

    #Inicializamos unas listas vacías
    corpus_plainText = list()
    corpus_verbatim = list()

    #Recorremos el corpus_content y lo dividimos en tres partes, el id, el␣
 ↪título y el texto
    for valor in corpus_content:
        partes = valor.split("\t")
        valor = partes[1].lower()
        document = {"id": partes[0], "title":partes[0].lower(),"text": valor}
        corpus_verbatim.append(document)
        corpus_plainText.append(valor)

    #Inicializamos el stemmer
    stemmer = Stemmer.Stemmer("english")
```

```
    #Tokenizamos el corpus
    corpus_tokenized = bm25s.tokenize(corpus_plainText, stopwords="en",␣
↪stemmer=stemmer, show_progress=True)

    #Creamos el índice BM25 y lo guardamos en un archivo
    retriever = createAndSaveRetriever(corpus_verbatim, corpus_tokenized)
```

La función anterior lee el archivo `train.docs` y lo divide en tres partes, la primera parte es el id[1] del documento, la segunda es el título del documento y la tercera parte es el texto del documento. Luego se tokeniza el texto del documento y se guarda en un índice BM25.

Este método llama a la función `createAndSaveRetriever()` que se encarga de crear el índice BM25 y guardarlo en un archivo.

1 El id es el mismo valor que el título presente en `train.docs`

La función `createAndSaveRetriever()` realiza lo siguiente:

1. Crea un retriever BM25 con el corpus verbatim y los métodos de cálculo de BM25 y de idf.
2. Indexa el corpus tokenizado.
3. Guarda el retriever en un archivo, para posteriormente ser recuperado

[ ]:
```
def createAndSaveRetriever(corpus_verbatim,␣
↪corpus_tokenized,method="lucene",idf_method="lucene"):

    #Creacion del retriever
    retriever = bm25s.BM25(corpus=corpus_verbatim, method=method,␣
↪idf_method=idf_method)
    #Indexado de la colección de documentos tokenizados
    retriever.index(corpus_tokenized, show_progress=True)

    #Guardado del retriever en un archivo
    retriever.save("../archivos/NFcorpus",corpus=corpus_verbatim)

    return retriever
```

En este documento también hay presente una función para poder cargar el retriever de un archivo, dicha función es `openRetriever()`:

[ ]:
```
def openRetriever():
    retriever = bm25s.BM25.load("../archivos/NFcorpus",load_corpus=True)
    return retriever
```

2 Ejercicio 2

En este segundo ejercicio se procede a cargar las queries y procesarlas con diferentes modelos de búsqueda para el retriever, y realizando dichas consultas con la combinación del steamer y

stopwords1,para posteriormente poder calcular la precisión de las consultas y guardarlas en un archivo.

1Se harán combinaciones de steamer y stopwords para poder comparar los resultados, dichas combinaciones son y para cada modelo de retriever: stopword-steaming, stopword-none-steaming, none-stopword-steaming, none-stopword-none-steaming.

Primero empezaremos cargando y preparando el corpus, para ello se utilizará la función `prepareCorpus()` y `createRetriever()` usadas en el ejercicio anterior y presentes en el archivo python del ejercicio 2. También se usará la función de `tokenizado()`, para poder tokenizar el corpus y las consultas según diferentes configuraciones

[ ]:
```
def tokenizado(texto, stemmer, stopwords):
    # Tokenizamos el corpus
    corpus_tokenized = bm25s.tokenize(texto, stopwords=stopwords,␣
 ↪stemmer=stemmer, show_progress=True)

    return corpus_tokenized
```

2.0.1 Tabla de resultados

Macro promedios

Modelo	Steamer	Stopwords	Precision	Recall	F1
Robertson	No	No	0.064	0.261	**0.101**
Robertson	No	Si	0.066	0.265	**0.103**
Robertson	Si	No	0.067	0.273	**0.105**
Robertson	Si	Si	0.069	0.279	**0.106**
Atire	No	No	0.064	0.261	**0.1**
Atire	No	Si	0.066	0.265	**0.102**
Atire	Si	No	0.067	0.273	**0.104**
Atire	Si	Si	0.069	0.278	**0.107**
Bm25l	No	No	0.064	0.263	**0.101**
Bm25l	No	Si	0.066	0.266	**0.102**
Bm25l	Si	No	0.068	0.275	**0.105**
Bm25l	Si	Si	0.069	0.279	**0.107**
Bm25+	No	No	0.064	0.261	**0.1**
Bm25+	No	Si	0.066	0.265	**0.102**
Bm25+	Si	No	0.067	0.273	**0.104**
Bm25+	Si	Si	0.069	0.278	**0.107**
Lucene	No	No	0.064	0.261	**0.1**
Lucene	No	Si	0.066	0.265	**0.102**
Lucene	Si	No	0.067	0.273	**0.104**
Lucene	Si	Si	0.069	0.278	**0.107**

Micro promedios

Modelo	Steamer	Stopwords	Precision	Recall	F1
Robertson	No	No	0.064	0.196	**0.097**
Robertson	No	Si	0.066	0.2	**0.099**
Robertson	Si	No	0.067	0.205	**0.101**
Robertson	Si	Si	0.069	0.21	**0.104**
Atire	No	No	0.064	0.196	**0.097**
Atire	No	Si	0.066	0.2	**0.099**
Atire	Si	No	0.067	0.205	**0.101**
Atire	Si	Si	0.069	0.21	**0.104**
Bm25l	No	No	0.064	0.197	**0.097**
Bm25l	No	Si	0.066	0.2	**0.099**
Bm25l	Si	No	0.068	0.206	**0.102**
Bm25l	Si	Si	0.069	0.21	**0.104**
Bm25+	No	No	0.064	0.196	**0.097**
Bm25+	No	Si	0.066	0.2	**0.099**
Bm25+	Si	No	0.067	0.205	**0.101**
Bm25+	Si	Si	0.069	0.21	**0.104**
Lucene	No	No	0.064	0.196	**0.097**
Lucene	No	Si	0.066	0.2	**0.099**
Lucene	Si	No	0.067	0.205	**0.101**
Lucene	Si	Si	0.069	0.21	**0.104**

Gracias a los datos obtenidos podemos concluir lo siguiente:

- Macro promedios -> se consigue un valor F1 de 0.107
  - Robertson con steamer y stopwords
  - Atire con steamer y stopwords
  - Lucene con steamer y stopwords
  - Bm25+ con steamer y stopwords
  - Bm25l con steamer y stopwords
- Micro promedios -> se consigue un valor F1 de 0.104
  - Atire con steamer y stopwords
  - Bm25l con steamer y stopwords
  - Robertson con steamer y stopwords
  - Bm25+ con steamer y stopwords
  - Lucene con steamer y stopwords

De los modelos que mejor se comportan podemos extraer que a términos de Macro promedios el mejor modelo es Bm25l y Robertson, ambos con steamer y stopwords, ya posee un mayor recall. En cuanto a los Micro promedios, no hay ninguno que destaque por encima de los demás.

Podemos concluir que Bm25l y Robertson son los modelos que mejor se comportan en valores de Macro y Micro, haciendo que cualquiera de los dos sea una buena elección para realizar consultas. Basándonos en estos resultados, el modelo con el cual se seguirá trabajando será Bm25l con steamer y stopwords.

3 Bibliografía

Enlaces consultado para la realización de los ejercicios:

3.1 Ejercicio 1

- Introduction to Lexical Search

3.2 Ejercicio 2

- Introduction to Lexical Search
- Lexical Search and Performance Evaluation