Proyecto Final de Sistemas de Recuperación de la Información

Informe

– Daniela Rodríguez Cepero Grupo: C311
– Belsai Arango Hernández Grupo: C311
– Carlos Carret Miranda Grupo: C312

Abstract. Currently Information Retrieval Systems are widely used in various areas, search engines use them constantly. This report describes each stage of the process carried out for the implementation of an Information Retrieval System. Details of tools will be covered used and most important aspects of the code.

Keywords: web, FND, FNDC,data base, Information Retrieval System

Resumen En la actualidad los Sistemas de Recuperación de Información son muy usados en diversas áreas, los motores de búsqueda los utilizan constantemente. En el presente informe se describe cada etapa del proceso llevado a cabo para la implementación de un Sistema de Recuperación de la Información. Se abordarán los detalles de las herramientas empleadas y aspectos mas importantes del código.

Palabras claves: web, FND, FNDC,bases de datos, Sistema de Recuperación de Información.

1 Introducción

La información es conocimiento sobre un determinado hecho o circunstancia.
La recuperación se refiere a la búsqueda a través de la información almacenada para encontrar información relevante. Ante esto, los sistemas de recuperación de información (SRI) se ocupan de la representación, el almacenamiento, la organización de/y el acceso a los elementos de la información. Los tipos de elementos de información incluyen documentos, páginas web, catálogos en línea, registros estructurados, objetos multimedia, entre otros. La función principal de los SRI son indexar texto y buscar documentos útiles en una colección para darle respuesta a las múltiples consultas de los usuarios.

2 Diseño

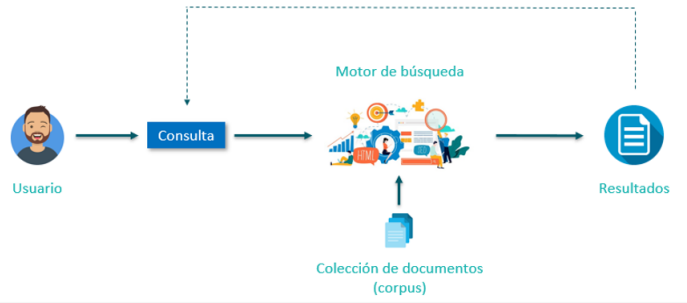

Figura 1. Diseño del SRI[1]

Para el funcionamiento del sistema se tiene una colección de documentos, los cuales son procesados para acceder a su información de manera mas sencilla.
El usuario introduce una consulta en forma de texto en el sistema de recuperación de información. Luego, el sistema procesa esta consulta y busca los documentos relevantes para el usuario de la colección que fue procesada. Una vez realizado este proceso, se le devuelven al usuario el título de estos documentos que fueron relevantes.

Para la realización de este proyecto se seleccionaron tres modelos de recuperación de información, para encontrar la similitud entre los documentos y las consultas realizadas por el usuario, dos ellos son clásicos(el boolenao y el vectorial) y el otro modelo fue el fuzzy.
Independientemenete del modelo el sistema primero realiza un proceso de parser determinado por la colección que se desea, como 20 Newsgroups o Cranfield . Por cada documento se tiene un identificador, el título y el texto. En el caso de las consultas, el identificador y la consulta como tal en un txt para el caso de uso de la base datos Cranfield quien cuenta con un conjunto de consultas de pruebas.

Una de las estrategias usadas es el procesamiento de datos de la forma más rigurosa posible. Este proceso lo iniciamos colocando todas la palabras en minúsculas. Luego, como el lenguaje en que se encuentra el corpus es Inglés, se eliminaron las contracciones que con frecuencia se emplean. Además, dentro de grandes volúmenes de textos es posible ver palabras mal escritas que contengan símbolos o dígitos ajenos a ellas, esto es tenido en cuenta por el programa, al igual que los espacios en blanco en exceso. También se pasa por un proceso que eliminación de todas aquellas palabras que no ofrecen información relevante(stopwords). Después de haber culminado todo el proceso el texto final es almacenado en

una estructura creada llamada Doc junto a su identificador y título, para evitar realizar esta operación cada vez que se inicie el programa.

2.1 Análisis del desarrollo del modelo boolenao

El modelo booleano fue escogido debido a las ventajas que representa. Modelo simple basado en conjuntos. Fácil de comprender e implementar. Consultas con precisión semántica. Aunque tambien sex consideraron las limitaciones que presenta: la no correspondencia parcial, lo que resulta poco flexible , la inexistencia de una noción de ordenación por relevancia debido a que estos valores son binarios y que no siempre es simple traducir una necesidad informacional a una expresión boolena.

Se utiliza la clase boolean model para implementar las propiedades y métodos de este modelo.
A modo resumen la clase recibe la consulta parceada de manera que tiene los términos y los operadores and, or y not que se encuentran de manera explícita en la redacción y que luego todos los términos que no están relacionados entre ellos por un operaddor se le aplica el operador and.

Uno de los métodos de la clase es el de similitud entre una consulta y un documento. Para ello el método similitud recibe la consulta y los términos que tiene cada documento a través de un diccionario y lo que hace es que recorre la consulta de izquierda a derecha aplicando los operadores de manera tal que: el operador or es la unión, el operador and es la intersección y el operador not es la diferencia. Lo que hace es que busca los documentos donde se encuentran los términos de la relación actual de manera independiente y establece la operación correspondiente en los documentos obteniendo los que son relevantes y esa información la guarda en una variable que luego trabaja ese resultado con el operador y el término que viene en la siguiente iteración.

2.2 Análisis del desarrollo del modelo vectorial

Para su elección se tuvo en cuenta que la consulta no tiene que coincidir con exactitud con un documento para ser considerado relevante a diferencia del modelo booleano, gracias a su esquema de ponderación y la estrategia de coincidencia parcial. Aunque está limitado por considerar a los términos indexados como independientes, lo cual no ocurre así en la realidad, donde hay correlación entre algunos de ellos. Dicha problemática requiere de enfoques más avanzados.

En el modelo vectorial se asigna un peso positivo no binario a cada término en los documentos, este es guardado dentro de la estructura doc como una propiedad del mismo, al igual que los términos indexados en las consultas.

Para determinar cuán relevante es un documento de acuerdo a una consulta, se utiliza el coseno del ángulo comprendido entre los vectores documentos y con-

sulta , al que se le denomina similitud, esto se empleó en la confección del método:

```
def similitud(self, query, doc):

    q_weights = self.q_weights
    doc_weights = self.doc_weights

    sum_weights_total=0
    sum_cuadrado_doc_weights=0
    sum_cuadrado_q_weights=0

    for term in query:
        if term in doc.term_frec:
            sum_weights_total += doc_weights[(term,doc)] * q_weights[term]
            sum_cuadrado_doc_weights += pow(doc_weights[(term,doc)], 2)
            sum_cuadrado_q_weights += pow(q_weights[term], 2)
        else:
            sum_weights_total += 0
            sum_cuadrado_doc_weights += 0
            sum_cuadrado_q_weights += pow(q_weights[term], 2)
    if sum_cuadrado_doc_weights==0 or sum_cuadrado_q_weights==0:
        return 0

    sim_doc_q=sum_weights_total / (math.sqrt(sum_cuadrado_doc_weights) * math.sqrt(sum_cuadrado_q_weights))
    return sim_doc_q
```

La ponderación de los documentos fue realizada empleando la frecuencia normalizada $tf_{i,j}$ del término t_i en el documento d_j cuya ecuación es $tf_{i,j} = \frac{freq_{i,j}}{maxFreq_{i,j}}$. El máximo se calcula sobre los términos del documento d_j. Si el término t_i no aparece en el documento d_j entonces directamente $f_{i,j}$ es cero. La frecuencia normalizada se hace con el objetivo de evitar valores de frecuencia sesgados por la longitud del documento. Otra métrica utilizada para definir el peso de un término en un documento fue la frecuencia de ocurrencia del término t_i dentro de todos los documentos de la colección. Este se identifica como idf_i, que es igual a $\log \frac{N}{n_i}$, siendo N la cantidad de documentos en el sistema y n_i el total de documentos en que aparece t_i. Finalmente $w_{i,j} = tf_{i,j} * idf_i$.

```
def idf(self,query):
    idf_i={}
    docs_count=len(self.documents)
    for term in query:
        if term in self.tokens_list:
            idf_i[term]=math.log(docs_count/len(self.tokens_list[term]),10)
        else:
            idf_i[term]=0
    return idf_i
```

```
def docs_weights(self, query):

    freq_nor = self.freq_normal(query)
    idf_i = self.idf(query)

    doc_weights={}
    for term,doc in freq_nor.keys():
        doc_weights[(term,doc)]=freq_nor[(term,doc)] * idf_i[term]
    return doc_weights
```

En cuanto a las consultas el peso es igual a $w_{i,q} = (a + (1 - a)\frac{freq_{i,q}}{maxFreq_{i,q}}) * \log \frac{N}{n+i}$, donde $freq_{i,q}$ es la frecuencia del término t_i en el texto de la consulta q. Se utiliza a para amortiguar la contribución de la frecuencia del término, en el sistema utilizamos un valor de $a = 0{,}5$.

```
def query_weights(self, query):

    freq_nor_q = self.freq_normal_q(query)
    idf_i = self.idf(query)

    q_weights={}
    for term in query:
        q_weights[term]=( 0.5 + (1 - 0.5) * freq_nor_q[term] ) * idf_i[term]

    return q_weights
```

Cada uno de los pesos obtenidos para los documentos son almacenados para evitar operaciones innecesarias cada vez que cargue el sistema.

La estructura de datos empleada para gestionar de forma más eficiente el almacenamiento y la accesibilidad a los términos, fue índices invertidos, simulado mediante un diccionario. Cada término contiene a una lista de los documentos en donde está incluido.

```
class Vector_Model(Model):
    def __init__(self,documents):
        self.documents=documents
        self.tokens_list = self.load_documents(documents)
        self.q_weights=None
        self.doc_weights=None

    def load_documents(self, documents)-> dict:
        tokens_list = {}

        for doc in documents:
            tokens = set(doc.term)
            for token in tokens:
                if not token in tokens_list:
                    tokens_list[token] = [doc]
                else:
                    tokens_list[token].append(doc)

        return tokens_list
```

Para dar respuesta a una consulta, luego de pasar por todos los procesos anteriores, se diseñó un algoritmo que muestra los documentos relevantes en orden decreciente con respecto a la similitud, además se fija un umbral de valor igual a 0.1, donde los documentos resultantes deben tener una similitud superior.

```
if model =="vect":
    modelo.load_query(query)
    print(query)
    print("vect")
    similitud_dic ={}
    for doc in modelo.documents:
        similitud = modelo.similitud(query, doc)
        if(similitud >0.1):
            similitud_dic[doc] = similitud
            titles[doc.title]=dir_docs[doc.id]
```

2.3 Análisis del desarrollo del modelo fuzzy

El modelo Fuzzy intenta solucionar los problemas que presenta el modelo booleano como es el grado de relevancia entre los términos de la consulta y los documentos y la correlación entre los términos indexados, aplicando la lógica difusa. De forma general estas fueron las causas por la cual elegimos este modelo como último modelo a desarrollar.
Para la comprensión del documento y la consulta se realiza de la misma manera que en el modelo booleano, es decir en los documentos a partir de un vector de términos indexados, donde en la componente i-ésima aparecerá 1 si se encuentra el término en el documento y 0 en caso contrario. Las consultas tienen además de los términos indexados los operadores AND, OR y NOT, donde se trabaja llevando a su **Forma Normal Disyuntiva** y luego a su **Forma Normal Disyuntiva Completa**.

Tranformación de la consulta a Forma Normal Disyuntiva Apoyandonos de la librería *sympy* que se encuentra en Python donde existen varios métodos como son: *simplify* y *todnf*, los cuales se utilizan para simplificar una expresión booleana y para obtener una expresión en FND a partir de una expresión booleana respectivamente.
Estos métodos trabajan de forma diferente la notación de los operadores booleanos, por tanto como recibimos la consulta con operadores de la forma And,Or y Not se tuvo que realizar el cambio de notación donde:
And $\rightarrow$ &
Or $\rightarrow$ |
Not $\rightarrow \sim$

Tranformación de la consulta a Forma Normal Disyuntiva Completa Luego de tener una fórmula expresada en forma normal disyuntiva aplicando transformaciones equivalentes esta puede ser llevada a FNDC. La idea es agregar expresiones de la forma $a \vee \neg a$ si en la componente disyuntiva falta la variable

a.

Para llevar a cabo el algoritmo que realiza esta transformación se hace uso de las clases BooleanAlgOp y Component, donde la primera es la encargada de separar cada componente conjuntiva y la segunda es la que representa a una componente conjuntiva en particular.

```
@staticmethod
  def get_fndc(n_components: int, query: str, components_ref
      : list = None):
      """Dada una query, su cantidad de componentes y sus
          componentes devuelve un diccionario con las cc"""
      components_set = set(components_ref)     # Conjunto
          para tener todos los terminos sin repetir
      components_dict = BooleanAlgOp.create_components_dict(
          components_ref) # Diccionario con llaves: terminos
          , valores: indice en array. Al termino t1 le
          corresponde el indice 1 en el array
          components_refs
      component_list = BooleanAlgOp.extract_query_cc(query.
          split(), components_dict, components_set) # Aqui
          obtengo todas las CC y las agrego en una lista
          para luego quitar las que esten repetidas

      cc_final_list = []
      for component in component_list:
          for component_eval in component.eval():
              cc_final_list.append(component_eval)

      cc_final_dict = Component.reduce(cc_final_list,
          components_dict) # Aqui me deshago de las
          componentes repetidas

      return cc_final_dict
```

Primero se obtiene la lista con todas las variables que se encuentran en la consulta y se tendrán en cuenta en el proceso de conversión de FND a FNDC, el nombre de esta lista será *components_ref*. Luego se crea un diccionario para obtener el índice de cada término(variable) en la lista creada anteriormente haciendo uso del método **BooleanAlgOp.create_components_dict()**. Pasamos a la parte de separar cada componente conjuntiva, llamaremos *token* a todas las cadenas que forman parte de la consulta en FND, procedemos iterando por cada *token* en la consulta y creando una instancia de Component para representar a dicha componente en los siguientes casos:

- Se encuentra un | y por tanto se crea una Componente Conjuntiva con todo los *tokens* encontrado hasta el momento.

– Se llega al último token de la consulta y por tanto se crea una Componente Conjuntiva con todo los *tokens* encontrado hasta el momento.
– En caso de encontrarse con un (se asume que se comienza a crear una componente conjuntiva.

Luego de obtener una lista con todas las Componentes Conjuntivas, iteramos por cada instancia de Component de esta lista llamando al método:

Component.eval()

para obtener las Componentes Conjuntivas que se van formando al agregar las variables que no estaban en estas siguiendo la forma mencionada al comienzo de este apartado (agregar expresiones de la forma $a \vee \neg a$). Por último llamamos al método:

Component.reduce(components_list: list, str_components_dict: dict)

eliminando todas las Componentes Conjuntivas repetidas en nuestra FND y con esto ya obtendremos un diccionario donde cada llave es un string de la forma $(v_1 v_2 .... v_n)$ representando una expresión booleana como $(v_1 \wedge v_2 \wedge .... \wedge v_n)$, donde $v_i \in N$.Los valores de este diccionario serán cada Componente Conjuntiva que corresponde a la llave que le hace referencia.

Métodos del Modelo En cuanto a los métodos del modelo se utiliza una Matriz de Correlación para las relaciones existentes entre los términos indexados donde en nuestra implementación esta función recibe la consulta como parámetro y el procedimiento es el siguiente:
Cada fila y columna tiene asociado un término indexado y en la posicón i,j se almacena la relación que existe entre el término i y el término j que se calcula con el factor de correlación normalizado c_{ij} a través de la fórmula:
$\frac{n_{ij}}{n_i + n_j - nij}$
donde:
n_{ij}: cantidad de documentos que contienen ambos términos.
n_i: cantidad de documentos que contiene el término indexado k_i.
n_j: cantidad de documetos que contiende el térmnino indexado k_j.

En nuestro código para obtener n_{ij} se realiza la intersección entre los documentos que contienen k_i y k_j, en caso que el término no se encuentre en ningún documento entonces el valor de correlación con cualquier otro k en la matrix es 0.

```
def correlation_matrix(self, query):
    corr_matrix={}

    for term_i in query:
        for term_j in query:
            if term_i in self.tokens_list and term_j in self
                .tokens_list:
                n_i_j = len(set(self.tokens_list[term_i]).
                    intersection(set(self.tokens_list[term_j
                    ])))
                n_i = len(self.tokens_list[term_i])
                n_j = len(self.tokens_list[term_j])
                corr_matrix[(term_i,term_j)] = self.truncate
                    (n_i_j / (n_i + n_j - n_i_j),5)
            else:
                corr_matrix[(term_i,term_j)] = 0

    return corr_matrix
```

Auxiliándose de la correlación entre los términos se define un conjunto difuso por cada k_i dando lugar a que cada documento tenga un grado de pertenencia teniendo en cuenta los términos indexados que contiene y se calulca de esta manera:
$\mu_{ij} = 1 - \prod_{k_l \in d_j}(1 - c_{il})$
donde:
i es el índice de los términos indexados.
j es el índice de los documentos.

En nuestra implementación por cada documentos buscamos los términos de la consulta que se encuentran y aplicando la fórmula multiplicamos cada correlación existente entre esos términos obteniendo el grado de pertenencia correspondiente, el cual si todos los índices del documento d_j están débilmente relacionados con k_i entonces k_i no es un buen índice difuso para el d_j.

```
def fuzzy_set_associated_with_doc(self):
    fuzzy_doc={}

    for doc in self.documents:
        for i in range(0,len(self.query_term)):
            term=self.query_term[i]
            if not (term,doc) in fuzzy_doc.keys():
                mult_kterm_doc=1
                for k in range(0,len(self.query_term)):
                    if self.query_term[k] in doc.term_frec:
                        mult_kterm_doc*=self.truncate(1 -
                            self.correl_matrix[(term,self.
                            query_term[k])],5)

                fuzzy_doc[(term,doc)]=self.truncate(1 -
                    mult_kterm_doc,5)

    return fuzzy_doc
```

El objetivo principal de estos métodos descritos anteriormente es para calcular la función ranking que tiene un documento d_j, de esta manera se obtiene el conocimiento de que tan relevante es con respecto a una consulta q, siguiendo la expresión siguiente:
$\mu_{qj} = 1 - \prod_{i=1}^{p}(1 - \mu_{cc_{ij}})$
p: cantidad de componentes conjuntivas de q.
$\mu_{cc_{ij}}$: grado de pertenencia d_j a la componente conjuntiva i-ésima de q
$\mu_{cc_{ij}} = \prod_{l=1}^{n} t_l$
t_l es μ_{ij} si el término l es positivo en la componente conjuntiva actual, en caso de que sea negativo sería $1 - \mu_{ij}$

En el caso de la implementación nuestro método recibe un documento y realiza el calculo aplicando la fórmula antes descrita donde para saber si el término actual es un literal positivo o negativo contamos con un vector de términos binarios que nos brinda esa información para cada componente conjuntiva de la Forma Normal Disyuntiva Completa.

```
def ranking_function(self, doc):
    index_literal=self.class_BooleanAlgOp.components_dict

    cc_p=1
    for literal in self.q_fndc.keys():

        literal_list = literal.split("_")
        miu_cc_i_j=1
        for item,index in index_literal.items():
            if literal_list[index]:
```

```
                miu_cc_i_j =self.truncate(miu_cc_i_j, 5) *
                    self.mui_i_j[(item,doc)]
            else:
                miu_cc_i_j = self.truncate(miu_cc_i_j, 5) *
                    (1 - self.mui_i_j[(item,doc)])
        cc_p *= float((1 - self.truncate(miu_cc_i_j,5)))

    return float(1-self.truncate(cc_p, 3))
```

3 Herramientas de desarrollo

Utilizamos el lenguaje de programación python 3.10

Para la creación de la api se utilizaron las librerías: django, djangorestframework, djangorestframework-simplejwt, django-model-utils, django-cors-headers.

Para simplificar una expresión booleana y para obtener una expresión en FND a partir de una expresión booleana respectivamente nos apoyamos en la librería *sympy* de Python donde existen varios métodos como son: *simplify* y *todnf*, los cuales se utilizan

Los datos fueron procesados con la ayuda de la librería nltk en la clase Parser. Esta nos ayudó a la separación de los textos de la consulta y los documentos en términos y de ella obtuvimos los stopwords que luego fueron removidos en los textos procesados.

La expansión en las consultas mediante la sustitución de las contracciones fue realizada con ayuda de la biblioteca re.

El trabajo vectorial se gestiona con el uso de la librería *numpy* y *math*, por la comodidad que ofrecen en el trabajo algebraico y su eficiencia computacional.

4 Evaluación y conclusiones del sistema

El sistema de recuperación de información fue evaluado con las métricas de Precisión, Recall y F_1.

La precisión esta dada por $\frac{|RR|}{|RR \cup RI|}$. En el caso de RR representa a los documentos relevantes recuperados y RI a los documentos irrelevantes recuperados. La precisión usualmente tiende a decrecer cuando la cantidad de documentos recuperados aumenta.

```
def Precision(self,cran_querys,dict_querys):
    precision  = []
    Relevantes_q = []
    for q in dict_querys.keys():
        Relevantes_q = cran_querys[q].split()
        Recuperados = dict_querys[q].__len__()
        RRecuperados_q= intersection(Relevantes_q,dict_querys[q])
        precision.append(RRecuperados_q/Recuperados)

    return precision
```

```
def Precision(self,cran_querys,dict_querys):
    precision  = []
    Relevantes_q = []
    for q in dict_querys.keys():
        Relevantes_q = cran_querys[q].split()
        Recuperados = dict_querys[q].__len__()
        RRecuperados_q= intersection(Relevantes_q,dict_querys[q])
        if RRecuperados_q and Recuperados:
            precision.append(RRecuperados_q/Recuperados)
        else:
            precision.append(0)
    return precision
```

El recall o recobrado es una función no decreciente para el número de documentos recuperados. Esta dada por $\frac{|RR|}{|RR \cup NR|}$, donde NR es el total de documentos relevantes no recuperados. Ahora, recuperando todos los documentos para todas las consultas se puede conseguir un Recobrado alto, pero la Precisión será baja.

```
def Recobrado (self,cran_querys,dict_querys):
    recobrado  = []
    Relevantes_q = []
    for q in dict_querys.keys():
        Relevantes_q = cran_querys[q].split()
        Recuperados = dict_querys[q]
        RRecuperados_q= intersection(Relevantes_q,Recuperados)
        NR = Relevantes_q.__len__() - RRecuperados_q
        recobrado.append(RRecuperados_q/(RRecuperados_q + NR))
    return recobrado
```

```
def Recobrado (self,cran_querys,dict_querys):
    recobrado  = []
    Relevantes_q = []
    for q in dict_querys.keys():
        Relevantes_q = cran_querys[q].split()
        Recuperados = dict_querys[q]
        RRecuperados_q= intersection(Relevantes_q,Recuperados)
        NR = Relevantes_q.__len__() - RRecuperados_q
        if RRecuperados_q and (RRecuperados_q + NR):
            recobrado.append(RRecuperados_q/(RRecuperados_q + NR))
        else:
            recobrado.append(0)
    return recobrado
```

Estas medidas se compensan, no están implicadas inversamente proporcionales. Lo ideal para los sistemas es obtener un recobrado de alto grado tolerando una pequeña cantidad de falsos positivos.

La medida F_1 es un caso particular de la medida F en la que la precisión y el recobrado sustentan igual importancia. Se define como $\frac{2}{\frac{1}{P}+\frac{1}{R}}$. Como se puede observar se emplea la media armónica, y así F_1 obtiene un alto valor cuando la Precisión y el Recobrado son ambos altos.

```
def f1(self,p, r):
    f1 = []
    for i in range(0,len(p)):
        pi,ri = 0,0
        if(p[i]!=0):
            pi = 1/p[i]
        if(r[i]!=0):
            ri = 1/r[i]
        d = pi + ri
        if d!= 0:
            f1.append( 2/d)
        else:
            f1.append(0.0)
    return f1
```

Dada las colecciones de consultas se obtiene el promedio de las métricas para distintas cantidades máximas de documentos relevantes, los resultados se muestran en lo adelante:

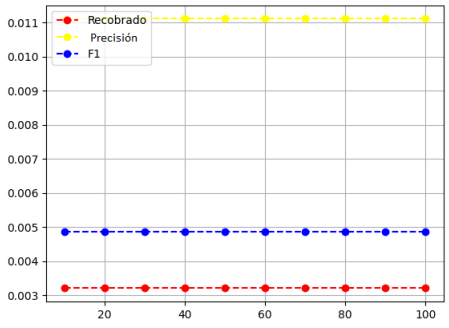

Figura 2. Medidas del modelo booleano con Cranfield

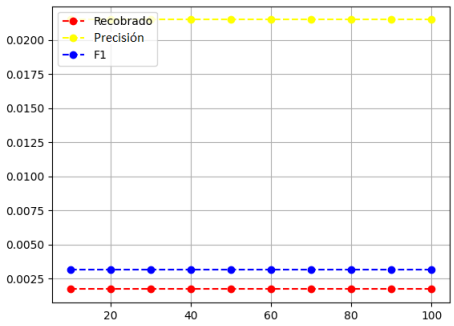

Figura 3. Medidas del modelo booleano con Vaswani

Como se puede apreciar en las gráficas 2 y 3 el comportamiento para ambas colecciones de documentos es similar. Además podemos ver como la precisión, el recobrado y F_1 se mantienen constantes debido a que el modelo recupera un número de documentos inferior a 10 para cada una de las querys de consulta de las colecciones por lo que el aumento de los documentos recuperados no influye en los valores de estos, y por tanto tampoco en el valor de F_1. Podemos ver que los valores que toman son extremadamente bajos por lo que el modelo representado tiene pésimos resultados.Con respecto a F_1 podemos ver como el valor que toma esta lejos de la precisión y más cerca del recobrado debido debido a sus bajos valores y que esta medida es un intento por mantener una armonía entre la precisión y el recobrado.

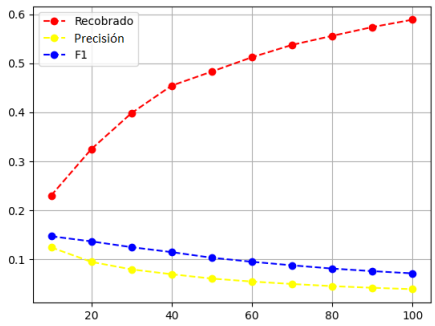

Figura 4. Medidas del modelo vectorial con Cranfield

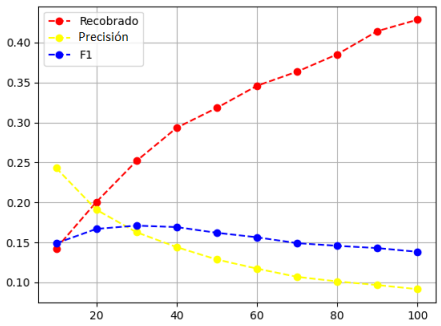

Figura 5. Medidas del modelo vectorial con Vaswani

Como se puede percibir en ambas gráficas se muestra un comportamiento similar, en donde se observa que a medida que aumenta la cantidad máxima de documentos que pueden devolverse como resultado de una consulta, también aumentan los valores de recobrado, pues resulta más probable que tengamos nuevos documentos relevantes al explorar en una vecindad mayor del total de documentos. Véase que para una cantidad máxima de 100 documentos posibles a devolver, se obtiene una gran cantidad de documentos relevantes en promedio de todas las consultas. En el caso de la precisión se tiene una tendencia a la disminución porque trae consigo un aumento de documentos no relevantes a la consulta en comparación a los relevantes. Dichas métricas mantienen un comportamiento opuesto a pesar de no ser magnitudes inversamente proporcionales. Teniendo en cuenta la métrica F_1 una buena práctica que se tuvo en cuenta en la Aplicación es devolver los 10 documentos con mayor similitud de acuerdo a cada consulta, ya que es donde mejor compensadas están el recall y la precisión. Trae consigo un resultado próximo a lo esperado por el usuario y propicia un bajo número de documentos que no resulten relevantes para él. Los valores decrecientes de F_1 son el resultado de lo alejado que se encuentran el recall y la precisión para cada muestra, que está afectada por un conjunto pequeño de documentos relevantes de las colecciones y una precisión no tan eficiente de los resultados.A pesar del comportamiento similar de las gáficas podemos ver como con cranfield 4 se obtiene un mayor recobrado para una misma cantidad máxima de documentos recuperados y para Vaswani 5 se obtiene un mayor valor de F_1 por lo que existe una mejor armonía con respecto a los valores que alcanza en otra colección.

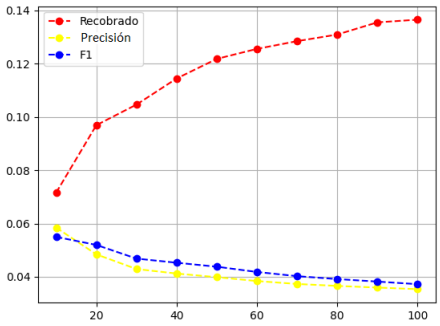

Figura 6. Medidas del modelo fuzzy con Cranfield

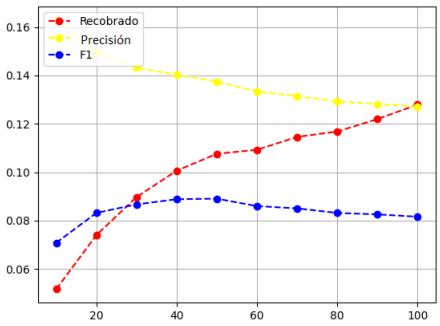

Figura 7. Medidas del modelo fuzzy con Vaswani

Como se puede percibir en ambas gráficas se muestra un comportamiento similar, en donde se observa que a medida que aumenta la cantidad máxima de documentos que pueden devolverse como resultado de una consulta, también aumentan los valores de recall; pues resulta más probable que tengamos nuevos documentos relevantes al explorar en una vecindad mayor del total de documentos. Véase que para una cantidad máxima de 100 documentos posibles a devolver, se obtiene una gran cantidad de documentos relevantes en promedio de todas las consultas. En el caso de la precisión se tiene una tendencia a la disminución porque trae consigo un aumento de documentos no relevantes a la consulta en comparación a los relevantes. Dichas métricas mantienen un comportamiento opuesto a pesar de no ser magnitudes inversamente proporcionales.Los valores decrecientes de F_1 son el resultado de lo alejado que se encuentran el recall y la precisión para cada muestra, que está afectada por un conjunto pequeño de

documentos relevantes de las colecciones y una precisión no tan eficiente de los resultados. Podemos observar que para la gráfica 7 en un inicio tienden a crecer los valores de F_1 a medida que aumenta la cantiad máxima de documentos a recuperar hasta un total máximo de 40 documentos, momento en que se alcanza el mayor valor para esta medida y por tanto un mayor equilibrio entre la precisión y el recobrado, luego comienza a disminuir. A pesar del comportamiento similar de las gáficas podemos ver como con 6 se obtiene un mayor recobrado para una misma cantidad máxima de documentos recuperados y para 7 se obtienen mayores valores de precisión y F_1 por lo que existe una mejor armonía con respecto a la otra colección. Por tanto podemos decir que el modelo fuzzy tuvo mejores resultados con la colección de Vaswani.

5 Ventajas y Desventajas

El sistema ofrece la posibilidad de extender las consultas más allá de las ofrecidas en las colecciones, proporcionándoles un manejo adecuado. Se cuenta con dos colecciones de datos Cran , 20 Newsgroups y Vaswani. Se le permite al usuario además de introducir una consulta la posibilidad de elegir dentro de las consultas de pruebas.
Como desventajas podemos citar, el no uso de machine-learning, que para sistemas avanzados les permite un mayor acercamiento a lo deseado por el usuario en su consulta aumentando en gran medida la precisión en las respuestas. También se consideran las desventajas de cada uno de los modelos mencionadas en el análisis de cada uno.

6 Recomendaciones

Recomendamos la incorporación de la retroalimentación a los modelos utilizados y uso de bases de conocimientos como ontologías o tesauros, puesto que favorecerán el desempeño de los modelos. También emplear algoritmos Crawling junto a colecciones que permitan su uso.

Referencias

1. Conferencias de SRI. Facultad de Matemática y Computación. Universidad de la Habana.

2. Wikipedia. URL: http://en.wikipedia.org. Consultado en 20 de diciembre de 2022.
3. Oliva Santos, R. Introducción a los Sistemas de Recuperación de Información, 2012-2013.4.

4. Baeza-Yates, R. y Ribeiro-Neto, B., Modern Information Retrieval I, octubre, 1998.6.National Center for Biotechnology Information,http://www.ncbi.nlm.nih.gov.

5. Sitio: eMathTeacher: Método de Mamdani de Inferencia Borrosa. Disponible en: http://www.dma.fi.upm.es/research/FundMatSoftComputing/fuzzyinf/main.htm.

6. García Garrido, L. Introducción a la Teoría de Conjuntos y a la Lógica. Cuba: 2002.

7. Torres Sánchez, Maité. Garrido Martínez, Lourdes. Modelo de Recuperación de Información Fuzzy. Matcom, UH. La Habana, Cuba.