Sistema de Recuperación de Información

Amalia N. Ibarra Rodríguez, Gabriela B. Martínez Giraldo, and Sandra Martos Llanes

Universidad de La Habana,
Cuba
{amalia.ibarra,gabriela.martinez,sandra.martos}@estudiantes.matcom.uh.cu

1 Introducción

El mundo moderno se encuentra en momentos de constante desarrollo, nuevos descubrimientos y aumento exponencial de la cantidad de información que se almacena, esta información se traduce en datos, y obtener los datos requeridos desde una gran cantidad diferentes de fuentes se ha convertido en una tarea crítica. Aquí entran los Sistemas de Recuperación de Información(SRI) que juegan un papel primordial en la recuperación de documentos que tengan valor para el usuarión, de la forma más eficiente y precisa posible.

Para este proyecto se desarrolló un sistema de recuperación de documentos que implementa dos modelos, el modelo vectorial y el modelo booleano, se evaluarán ambos utilizando distintos conjuntos de documentos con sus respectivas consultas, y se comprobará cuan buenos son a través de medidas de evaluación objetivas.

2 Diseño

El proyecto fue implementado en python y cuenta 4 etapas fundamentales:

1. Procesamiento y representación del texto
2. Modelo Vectorial
3. Modelo Booleano
4. Evaluación

Se implementan tres clases principales:Document, Query y Corpus, esta última representa el conjunto de documentos de una set de datos, por los que almacena una lista de instancias de la clase Document .

Las clases Query y Corpus cuentan con funciones para el procesamiento y representación del texto, así como utilidades para la implementación de los modelos, todo esto se explicará en las próximas secciones.

2.1 Procesamiento y representación

Se utiliza el módulo ir_datasets para cargar los conjuntos de datos utilizados. Primero se lleva cada elemento del conjunto de datos, documentos,queries, y

relevancia, a una representación cómoda: conjuntos de instancias de clases Document, Query, y luego un Corpus.

Las clases Query y Corpus conocen como procesar los datos, y aunque difieren en algunos aspectos la idea es la misma:

Se utilza la biblioteca re y nltk que contiene un conjunto de método útiles para el procesamiento del lenguaje natural, que cuenta con varias etapas.

1. **Tokenización:** Básicamente separa por espacios y elimina algunos detalles de escritura obteniendo una lista de tokens que representan todas las palabras y signos del texto.
2. **Stemming(Derivación):** El proceso de reducir una palabra a su formato de raíz gramatical.
3. **Lematización:** La transformación que usa un diccionario para mapear la variante de una palabra a su formato raíz gramatical
4. Eliminación de *stopwords*

Esto se sintetiza en los métodos: tokenize stemmize y lemmatize_ de las clases Query y Corpus, dejando como resultado el conjunto de términos indexados para los documentos y queries.

2.2 Modelo Vectorial

Las clases Query y Document poseen diccionarios que tienen información sobre la frecuencia de cada término en su query y documento respectivo, así como la máxima frecuencia.

Se implementan métodos para calcular la frecuencia normalizada(tf_{ij}), la frecuencia de un término en el corpus($idf_{i,j}$), la cantidad de documentos donde aparece un término i, (n_i). Se implementaron estos cálculos de acuerdo a las siguientes fórmulas:

$$tf_{ij} = \frac{freq_{ij}}{max_f req_j} \tag{1}$$

$$idf_{ij} = \log \frac{N}{n_i} \tag{2}$$

Estos valores se utilizaran luego para calcular el peso de un término sobre un documento(w_{ij}).

$$w_{ij} = tf_{ij} \times idf_{ij} \tag{3}$$

Para (1), (2), (3) y las siguientes ecuaciones, i representará un término y j un documento específico.

En una query para calcular la ponderación de sus términos se sigue la siguiente línea:

```
def  set_weight_values(self,document,corpus,alpha = 0.5):
        for term in document.terms_vector.keys():
            tf,idf = 0,0
            if term in self.terms_vector.keys():
                tf = self.terms_vector[term] / self.max_freq

                if term in corpus.n_i.keys():
                    idf = math.log(corpus.N/ corpus.n_i[term])

            w = (alpha + ((1-alpha)*tf))*idf
            self.weights.append(w)
```

Mientras que en el corpus:

```
def set_weight_values(self):
        for document in self.documents:
            for term in document.terms_vector.keys():
                tf = document.terms_vector[term] / document.max_freq
                idf = math.log(self.N/self.n_i[term])
                document.weights.append(tf*idf)
```

Luego se calcula la similitud de una query con cada documento de acuerdo a:

$$sim(d_j, q) = \frac{\sum_{i=1}^{n} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{n} w_{i,q}^2}}, \tag{4}$$

una vez hecho esto, se calcula la media de las similitudes y todo documento que posea una similitud por encima de la mitad de esta media es recuperado como relevante para dicha query.

2.3 Modelo Booleano

Un documento solo se considera relevante para una query si todos los términos de esta están presentes en el documento, por lo que se implementa de una forma bastante simple.

```
def recover(corpus,queries):
    recovered = {}
    for document in corpus.documents:
        for query in queries:
            qlen = len(query.terms_vector.keys())
            count = qlen
            for qterm in query.terms_vector.keys():
                if count < qlen:
                    break

                if qterm in document.terms_vector.keys():
```

```
                    count
                else:
                    count -= 1

            if count == qlen:
                try:
                    recovered[query.id].add(document.id)
                except:
                    recovered[query.id] = set()
                    recovered[query.id].add(document.id)
    return recovered
```

3 Evaluación

Para la evaluación de estos modelos se utilizaron los conjuntos de datos Cranfield y Vaswani y las siguientes medidas:

1. Precisión: Fracción de los documentos recobrados que son relevantes.
2. Recobrado: Fracción de los documentos relevantes que fueron recuperados.
3. Medida F: La Precisón y el Recobrado pueden ser a veces medidad contrarias, por lo que debe haber una forma de medirlos de manera integral. Se utiliza F como una media armónica de la tasa de precisión y la tasa de recobrado. En nuestro cálculo damos más importancia a la precisión haciendo $\beta = 1.2$
4. Medida F1: Es similar a F solo que le da igual importancia a la Precisión y al Recobrado, mientras mayor sea F1 la prueba es más exitosa.

Obteniendo los siguientes resultados:

Table 1. Cranfield

Modelo	Precisión	Recobrado	Medida F	Medida F1
Vectorial	0.004	0.010	0.53	0.30
Booleano	0.0	0.0	0.0	0.0

Table 2. Vaswani

Modelo	Precisión	Recobrado	Medida F	Medida F1
Vectorial	0.01	0.01	0.40	0.39
Booleano	0.0	0.0	0.0	0.0

Como se puede apreciar el modelo booleano devuelve terribles resultados, resultando las medidas en valor 0, esto es debido a que los documentos que se

recuperaron no coinciden con los que conocemos que son relevantes, por tanto el conjunto de documentos Relevantes Recuperados es vacío, y como de él depende el éxito de un SRI entonces podemos decir que el modelo Booleano es fallido al menos para los corpus analizados. Una mejora podrí ser flexibilizar nuestro criterio de selección de documentos, en lugar de recuperar los documentos con un 100% de coincidencia recuperarlos con un 50%

Por otro lado, el modelo vectorial arroja mejores resultados, sin embargo no son extremadamente buenos, creemos que perfeccionando el criterio de selección del umbral de similitud pueden mejorar significablemente.