Sistema de Recuperación de Información Vectorial Con GUI para 20 Newsgroups

José Gabriel Navarro Comabella, Jorge Daniel Valle Diaz

Abstract. A Multi-Platform Vector Information Retrieval System with Multi-Platform Graphic User Interface using 20 Newsgroups Data

Keywords: Machine Translation, Translation, Statistical Machine Translation, Rule Based Machine Translation, Traducción Automática, Traducción, Traducción Automática Estadística, Traducción Automática basada en reglas, Ontology, Ontología, Knowledge Bases, Bases de conocimiento, Advantages, Disadvantages, Ventajas, Desventajas, Evaluation, Evaluación, Applications, Aplicaciones

1 Introducción

El presente sistema permite la recuperación de información del conjunto de datos 20 Newsgroups. Para ello se utiliza la arquitectura cliente-servidor y el modelo vectorial. No se disponen de consultas preelaboradas, ni una definición de relevancia con respecto a estas, así que presentamos las nuestras propias para la evaluación del sistema.

1.1 Recuperación de Información

La Recuperación de Información es la localización de materiales (generalmente documentos) de naturaleza no estructurada (generalmente texto) para satisfacer una necesidad de información en una larga colección (generalmente almacenada en computadoras)[1]

2 Sistema

2.1 Arquitecturas utilizada

Se utilizó la arquitectura cliente servidor por lo que el sistema tiene 2 componentes principales:

- El Servidor: Programado usando ASP.Net Core 3.1, lo que permite sea multi-plataforma, mayormente probado en Windows
- El Cliente: Interfaz gráfica programada usando Flutter 2, que también permite sea multi-plataforma, mayormente probado en Android

2.2 Modelo utilizado:

El modelo utilizado fue el vectorial debido a que presenta la posibilidad de ordenar los resultados de las consultas por similitud a la consulta a diferencia del booleano y permite aproximar la recuperación de documentos que se aproximen a los requerimientos de consulta. No se utilizó el modelo probabilístico debido a la falta de consultas junto con sus resultados esperados en términos de relevancia y por preferencia del otro modelo.

El modelo tiene varios de los componentes habituales de estos sistemas:

1. Se toma cada documento y se divide en palabras, en minúsculas, tomando en cuenta algunas excepciones como siglas separadas por puntos, o palabras separadas por guiones.
2. Se eliminan palabras que se consideran entorpecen el proceso (stopwords)
3. Luego utilizando el módulo Porter2Stemmer[2] se procesaron las palabras para dejarlas en una aproximación a su primitiva (Stemming).
4. Luego se obtuvieron las repeticiones de la palabra en ese documento
5. A continuación se guarda en la Base de Datos
6. Luego a partir de dichas repeticiones se genera la relevancia por cada una de estas primitivas en sus documentos
7. Por cada consulta se obtiene su vector de relevancia y se aplica la función de similitud para hallar los documentos similares ordenados del más parecido al menos.
8. Al seleccionar el documento este es servido al cliente

2.3 Servidor:

El servidor como ya se mencionó se puede usar en Windows, Linux y MacOS. El requisito es tener instalado el SDK de Net Core 3.1, aunque el desarrollo se realizó en una computadora usando Windows. Este es el que realiza la mayoría de los procesos antes expuestos.

Primero hay que verificar no esté inializada una base de datos antigua. Para esto en el directorio del servidor verificamos la existencia la carpeta Migrations/ y del archivo app.db; en caso de existir los eliminamos.

Luego se corren los comandos:

```
$ dotnet ef migrations add Migracion
$ dotnet ef database update
```

Después se inicia el servidor:

```
$ dotnet run
```

El comando anterior también debe instalar las dependencias faltantes desde Internet.

Luego una vez esté funcionando el servidor se le debe hacer un query especificando de donde debe leer los documentos. Esta lectura demora varias horas. Sin embargo va imprimiendo los documentos que ha procesado, si se decide utilizar un subconjunto de los datos es posible interrumpir el funcionamiento del servidor y volverlo a iniciar sin consecuencias, solamente habiendo leído hasta donde imprimió en la salida.

```
GET http://{URL del servidor}:5000/api/document/Carga/
{Path Carpeta Documentos}
//Ejemplo
GET http://localhost:5000/api/document/Carga/20news-18828
```

Luego es necesario pedirle que genere los vectores de Relevancia de los documentos:

```
GET http://{URL del servidor}:5000/api/document/
Relevancia
//Ejemplo
GET http://localhost:5000/api/document/Relevancia
```

Ya el servidor está listo para recibir consultas. Para ello responde a peticiones de la forma:

```
GET http://{URL del servidor}:5000/api/document/
Relevancia/Documentos/{Consulta}
//Ejemplo
```

```
http://localhost:5000/api/document/Relevancia/Documentos/
foot out of your mouth
```

Esto devuelve una lista de descriptores de documentos, que incluyen el id en la base de datos, el campo From y el campo Subject. Dicha respuesta se da en JSON.

A partir del id del documento es posible pedir el documento correspondiente a un id de la base de datos usando:

```
GET http://{URL del servidor}:5001/api/document/{id}
//Ejemplo
GET http://localhost:5000/api/document/700
```

2.4 Cliente:

El cliente se compone de una GUI implementada en Flutter que debería poderse compilar a Android, IOS y Web, incluso a versión de escritorio, aunque esta última se encuentra en Technical Preview. El desarrollo se realizó en Android, que requiere la previa instalación de Android Studio. Para compilar a distintas plataformas requiere distintas SDKs que se descargan automáticamente al intentar correrlo en la plataforma destino.

La GUI se compone principalmente de dos páginas:

1. La página principal que permite la realización de consultas y visualizar sus resultados y está compuesta por:
    a. Formulario de texto para realizar la consulta
    b. Lista de resultados
    c. Botón extra de búsqueda
2. La página de visualización de documentos donde se visualiza el documento presionado de la lista de resultados
    a. Texto del documento
    b. Botón superior para volver atrás
    c. Botón flotante para volver atrás

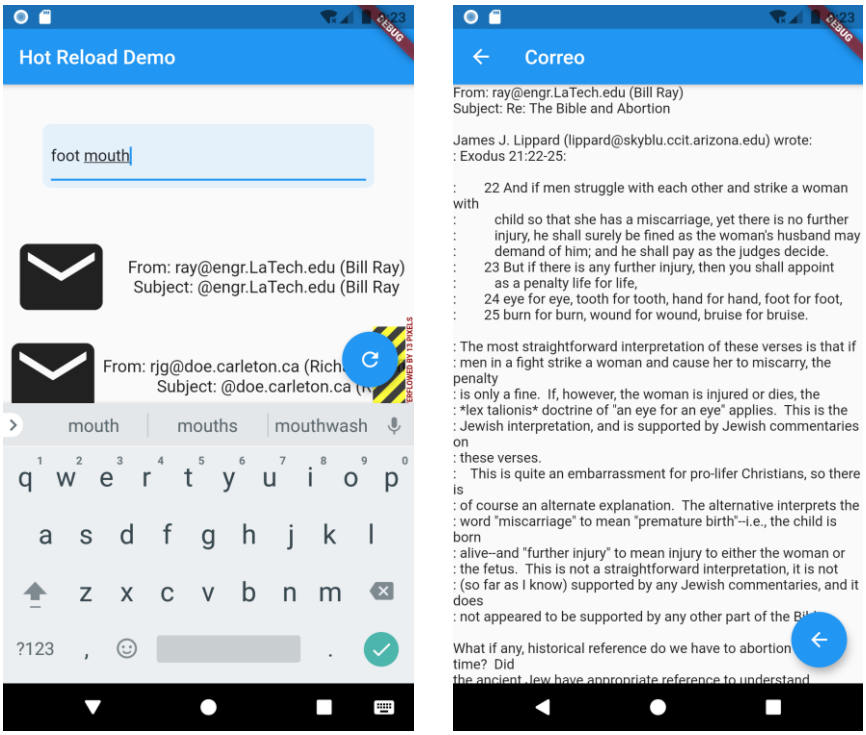

Fig. 1. Página principal y la de visualización

Para correr la aplicación es necesario tener instalado Flutter, abrir la carpeta donde se encuentra, escribir el siguiente comando:

```
$ flutter run
```

Y a continuación elegir un dispositivo compatible de los que esté conectado a la PC. Preferiblemente un dispositivo Android que es en lo que se probó la aplicación.

En la app se encuentra hard-coded el URL del servidor, para cambiarlo conviene revisar la línea 120 y 140 de lib/main.dart

3 Proceso de evaluación

Para la evaluación al no disponer de consultas y el conjunto de documentos relevantes relacionados se tuvo que desarrollar métodos propios que fueran asequibles para un no experto.

El método elegido fue seleccionar frases que parecieran específicas a un documento y luego evaluar la posición del documento en el ranking y si aparece dicho documento en las respuestas(Recall). Sumando esto a cierta revisión manual para asegurar no existan incongruencias mayores.

Las consultas utilizadas se adjuntan en el archivo Docs/Queries.txt en el par consulta terminado por id del documento origen.

Los resultados finales fueron en general satisfactorios. El documento del que se extrajo la frase no solía estar en posiciones muy alejadas del inicio aunque varias veces tampoco se encontraba entre los 5 primeros. No se observaron incongruencias en el resto de los documentos elegidos.

4 Resultados, Recomendaciones, Ventajas y Desventajas

El modelo presentado cumple su objetivo de una forma adecuada. Los objetos que devuelve sí suelen estar relacionados con la consulta realizada y presenta todas las ventajas propias del modelo vectorial mencionadas con anterioridad, además de que es fácilmente extensible. Aunque analiza las palabras como entidades individuales no relacionadas. Aparentemente también podría optimizarse para procesar las consultas más rápidamente, puesto que, aunque no es demasiado lento, se demora unos segundos en procesar la consulta. Se hizo el intento de agregar cierto Clustering pero se fracasó debido a problemas principalmente de rendimiento.

El sistema también se podría mejorar si se desarrollara un sistema de queries que permitan evaluarlo mejor y desarrollar cierta retroalimentación. Realizar relaciones de similitud entre documentos también podría ser útil para la usabilidad del sistema.

Referencias

1. Manning, C. D., *Introduction to Information Retrieval*, p. 1. (2009)
2. *Porter2-stemmer*, https://github.com/nemec/porter2-stemmer