Implementación de sistemas de recuperación de información basados en el estadístico $tf - idf$

Victor Manuel Cardentey Fundora
Grupo C511

Karla Olivera Hernández
Grupo C511

Amanda González Borrell
Grupo C511

Tutor(es):
Prof. Marcel E. Sánchez Aguilar

1. Introducción

En la actualidad el mundo se encuentra en un continuo desarrollo, cada día crece por parte de los investigadores el anhelo por buscar y descubrir conocimiento. Con los avances tecnológicos existentes; en su gran mayoría, este conocimiento se muestra y se guarda en forma de datos por lo que a medida que trascienden los días es mucho mayor la necesidad de contar con estructuras que de manera eficiente logren servir como herramienta de uso para la manipulación de tantos datos. En el presente informe se propone la implementación de un Sistema de Recuperación de Información, el cual permitirá recuperar documentos determinados por el relevancia que possean los mismos para el usuario. Esta relevancia estará dada por el estadístico $tf - idf$ el cual permite modelar la importancia de un término dentro de un documento perteneciente a un corpus determinado. Esta medida crece proporcionalmente con el número de veces que aparece el término en el documento y es ajustada de acuerdo a la cantidad de documentos del corpus donde aparece el término, esto permite capturar el hecho de que algunas palabras aparecen con más frecuencia que otras de forma general. Su facilidad de cómputo, la efectividad en la modelación de la importancia de términos y que pueda ser adaptada tanto a modelos vectoriales como probabilísticos hace que sea una de las medidas más utilizadas en la implementación de sistemas de recuperación de información. En este trabajo se implementará una clase base que permitirá implementar distintos modelos que utilicen esta medida, para ello brindaremos como ejemplo la implementación del modelo vectorial y del modelo probabilístico frecuentista Okapi BM25.

2. Desarrollo

2.1 Consideraciones sobre $tf - idf$

En esta sección se exponen los motivos que llevaron a seleccionar el modelo $tf - idf$ para la descripción de documentos.

Las ventajas consideradas fueron:

1. Es fácil de computar.
2. Permite describir los términos más relevantes de un documento de forma sencilla, por este motivo existen heurísticas como *skip lists* que permiten mejorar la eficiencia de los sistemas descartando términos poco prometedores.
3. Permite computar la similaridad de documentos de forma sencilla esto habilita la utilización de técnicas como clústering para mejorar los resultados del sistema.

Las desventajas consideradas fueron:

1. Basado en el modelo *Bag of Words* por lo que no captura información de la posición en el texto, coocurrencias en distintos documentos.
2. No puede capturar información semántica a diferencia de otros modelos como *word embeddings*.

Debido a estos hechos, remarcando principalmente la facilidad de cómputo y a que los modelos basados en $tf-idf$ han mostrado buenos resultados en la literatura se decidió utilizar este modelo.

2.2 Arquitectura

El sistema desarrollado se compone de 4 módulos:

1. Procesamiento de lenguaje natural
2. Modelos
3. Sistemas de Recuperación de Información
4. Interfaz Visual

El desarrollo de la solución se realizó en el lenguaje **python**

2.3 Procesamiento de texto

El primer paso en el *pipeline* de recuperación es transformar la representación en string del texto en un conjunto de términos significativos. Para esto se emplearon las bibliotecas **nltk**, **re**, **contractions**, **unicode** las cuales proveen un conjunto de métodos útiles para el procesamiento de lenguaje natural.

Se definió una interfaz para la funcionalidad de un procesador de texto que en este caso normaliza un string y lo convierte en un array de strings donde cada uno representa un término.

```
class TextNormalizer:
  def normalize(self, text):
    raise NotImplementedError()
```

Figura 1: Interfaz de procesador de texto.

La implementación provista en esta solución provee las siguientes funcionalidades:

1. Eliminación de espacios en blanco
2. Conversión de formato unicode a ascii
3. Expansión de contracciones gramaticales
4. Eliminación de símbolos de puntuación
5. Tokenización
6. Eliminación de *stopwords*
7. *Lemmatization*

2.4 Modelo basado en $tf - id$

La clase base **FrequencyModel** implementada en la solución permite el computo de estructuras de datos y medidas relevantes para modelos que utilizan este estadístico mediante el método *fit*

Entre las estructuras relevantes y datos sobre el corpus utilizados se encuentran:

1. Tabla tf: debido a que las tablas tf generalmente son esparcidas la implmentación se realizó utilizando un índice invertido el cual asocia a cada término t_i con pares (d_j, f_{ij}) manteniendo un acceso eficiente mediante el uso de diccionarios y reduciendo el costo de memoria de almacenar dicha tabla.
2. Tabla *df*: la cantidad de documentos en las que está presente un término es igual a $\textbf{len}(tf[t_i])$.
3. Tabla *idf*: se utiliza un diccionario para implementar esta tabla, sin embargo el cálculo de *idf* se definió como un método abstracto debido a que existen múltiples formas de calcular *idf*.
4. Tabla *doc_len*: asocia cada documento con la cantidad de términos distintos que aparecen en él.
5. Longitud media de un documento.
6. Cardinalidad del corpus.

Además provee múltiples definiciones de tf normalizado y de *idf* para poder utilizarse en clases herederas.

Luego define de forma genérica el calculo de similitud de un documento dividiendo este en dos funciones: *dscore* u *qscore*. *dscore(term, doc) -¿ float* retorna el peso del termino en el documento y *qscore(query)* debido a que el peso de los términos depende de la query este método devuelve una función para calcularlos de forma que *qscore(query) -¿ func(term) -¿ float*. Luego el proceso de calcular la similitud se reduce a multiplicar *dscore(term, doc) * func(term)*.

El proceso de *scoring* y recuperación de documentos también se definió de forma genérica y la clase provee dos estrategias de recuperación: *Document-At-A-Time* (DAAT) y *Term-At-A-Time* (TAAT)

```
for every query term q in Q do
  get indexed documents at tf table
  for each document d indexed
    score(d) = score(d) + weight(q,d)
normalize scores
sort documents
```

Figura 2: Estrategia TAAT.

```
for every document d in D do:
  for every query term q in Q:
    access inverted index in q,d
    score(d) = score(d) + weight(q,d)
  normalize score(d)
sort documents
```

Figura 3: Estrategia TAAT.

Estas formas de evaluaciones recuperan los k documentos con mejor puntuación. La estrategia TAAT usualmente es preferible con corpus pequeños mientras que DAAT obtiene mejores resultados con corpus de mayor tamaño.

Con estas definiciones se tiene un modelo genérico en el cual solo restaría definir los métodos para calcular los pesos.

2.5 Modelo vectorial

El modelo vectorial implementado se realizó de acuerdo a lo definido en las conferencias impartidas durante la asignatura.

Por lo que podeos definir *dscore* como:

$$w(t,d) = \frac{tf(t,d) \times idf(t)}{|d|}$$

Y *qscore* como:

$$w(t,q) = 1 + (1-a)\frac{tf(t,q) \times idf(t)}{|q|}$$

2.6 Modelo Okapi

El modelo Okapi es un modelo probabilístico el cual utiliza la frecuencia de términos en lugar del modelo de representación binaria. Se sugiere este modelo debido a que permite la implementación de esquemas de retroalimentación de ambos modelos, tanto clustering como enfoques de retroalimentación para la optimación de los parámetros del modelo. Por lo que se especula que con esquemas de retroalimentación se alcancen mejores resultados que el modelo vectorial.

1. El cálculo de *idf* se realiza mediante la fórmula $log(\frac{N}{n_i})$
2. La función de similitud utilizada es $\sum_{t \in q} \log(\frac{N}{n_t}) \frac{(k_1+1)tf_{td}}{k_1((1-b)+b(L_d/L_ave))+tf_{td}} * \frac{(k_3+1)tf_{tq}}{k_3+tf_{tq}}$ donde k_1, k_3 y b son parámetros de estimación, L_d es la cantidad de términos del documento y L_{ave} es la media de términos por documento en el corpus.

Por tanto podemos definir *dscore* como:

$$\log(\frac{N}{n_t}) \frac{(k_1+1)tf_{td}}{k_1((1-b)+b(L_d/L_ave))+tf_{td}}$$

Y *qscore* como:

$$\frac{(k_3+1)tf_{tq}}{k_3+tf_{tq}}$$

2.7 Retroalimentación

Luego de obtener un primer conjunto de documentos relevantes mediante uno de los modelos mencionados se le permite refinar dicho cunjunto marcando de manera visual aquellos que el usuario considere importantes, a partir de este conjunto y según el modelo sucede lo siguiente:
En el modelo Vectorial la retroalimientación se realiza mediante el Algoritmo de Rocchio el cual transforma el vector de pesos de la query inicial q_0 en uno nuevo que se acerque al conjunto de documentos que el usuario marcó como relevantes y con esta nueva query se obtiene un nuevo conjunto de documentos recuperados final.
La fórmula para obtener la nueva query q_m:

$$\vec{q_m} = \alpha \vec{q_0} + \frac{\beta}{|D_r|} \sum_{\vec{d_j} \in D_r} \vec{d_j} - \frac{\gamma}{|D_{nr}|} \sum_{\vec{d_j} \in D_{nr}} \vec{d_j}$$

D_r y D_{nr} conjunto de documentos relevantes y no relevantes a partir de los documentos recuperados.
α, β y γ pesos establecidos para cada término de la consulta.
En el modelo Okapi al igual que en el vectorial se obtiene un primer conjunto de documentos recuperados del cual el usuario divide en dos conjuntos marcando los relevantes. Luego se recomputan los valores de los vectores de peso de los documentos teniendo en cuenta la cardinalidad de estos conjuntos y de otros parámtetros, el score o similitud total de un documento se obtiene en la retroalimentación mediante la siguiente fórmula:

$$\sum_{t \in q} log[[\frac{(|VR_t| + \frac{1}{2})/(|VNR_t| + \frac{1}{2})}{(df_t - |VR_t| + \frac{1}{2})/(N - df_t - |VR| + |VR_t| + \frac{1}{2})}]$$
$$* \frac{(k_1+1)tf_{td}}{k_1((1-b)+b(L_d/L_{ave}))+tf_{td}} * \frac{(k_3+1)tf_{tq}}{k_3+tf_{tq}}]$$

donde VR_t es el conjunto de documentos recuperados relevantes que contienen al término x_t, VR el conjunto de documentos relevantes conocidos, VNR_t el conjunto de documentos recuperados no relevantes que contienen al término x_t, df_t el número de documentos que contienen al término x_t y, por último, k_1, k_3 y b son parámetros de estimación.

2.8 Corpus y Evaluación

Para probar los modelos se utilizaron los corpus Cranfield, Vaswani y TrecCovid1 , estos fueron obtenidos a través de la API de **ir_datasets** la cual provee acceso a colecciones de documentos utilizados para la evaluación de sistemas de recuperación y otras áreas de NLP. A continuación mostramos las distintas medidas objetivas utilizadas:

1. P: Precisión
2. R: Recall
3. Success: Porciento de consultas en las que fue recuperado al menos un documento relevante
4. Rprec: R-Precisión

Los resultados fueron calculados para los Top 5 y Top 10 documentos por consulta debido a que los usuarios raramente exploran más documentos si los documentos previos no fueron de utilidad para ellos.

Modelo	P	R	Success	Rprec
Vectorial	0.006	0.001	0.01	0.004
OkapiBM25	0.007	0.001	0.013	0.003

Figura 4: Resultados Cranfield($k = 5$)

Modelo	P	R	Success	Rprec
Vectorial	0.3	0.1	0.7	0.16
OkapiBM25	0.4	0.13	0.8	0.214

Figura 5: Resultados Vaswani($k = 5$)

Modelo	P	R	Success	Rprec
Vectorial	0.4	0.03	0.966	0.088
OkapiBM25	0.61	0.05	0.966	0.11

Figura 6: Resultados TrecCovid($k = 5$)

Como podemos observar el modelo OkapiBM25 supera al modelo vectorial en todos los escenarios. El dataset Cranfield mostró notables problemas para ambos modelos con muy bajos resultados en todas las medidas, esto se pude deber a la poca cantidad de queries en

Modelo	P	R	Success	Rprec
Vectorial	0.005	0.003	0.02	0.003
OkapiBM25	0.006	0.005	0.05	0.004

Figura 7: Resultados Cranfield($k = 10$)

Modelo	P	R	Success	Rprec
Vectorial	0.23	0.15	0.79	0.16
OkapiBM25	0.34	0.2	0.9	0.214

Figura 8: Resultados Vaswani($k = 10$)

Modelo	P	R	Success	Rprec
Vectorial	0.4	0.06	0.93	0.08
OkapiBM25	0.56	0.06	1.0	0.11

Figura 9: Resultados TrecCovid($k = 10$)

el dataset y la poca cantidad de documentos relevantes por query. Los datasets Vaswani y TrecCovid mostraron lo que se han considerado buenos resultados dado que en los primeros 5 documentos recuperados existen 2 los cuales son relevantes de acuerdo a los valores de precisión obtenidos por lo cual el usuario debería de poder obtener información en su revisión inicial y los valores para los 10 primeros resultados mantiene una buena precisión lo que asegura entre 4 y 5 documentos relevantes en la primera página de recomendación de nuestro sistema.

3. Conclusiones

En este trabajo se ha realizado la implementación de un modelo abstracto de recuperación de información que permita implementar múltiples modelos basados en la utilización de medidas $tf - idf$ para ello se ha provisto una solución utilizando Streamlit para crear una interfaz de usuario amigable que facilite la experimentación y la implementación del modelo vectorial y okapi. Por otro lado, se creó un módulo de evaluación que permite realizar experimentos sobre estos modelos utilizando los corpus provistos por la API de **ir_datasets** lo cual permita determinar cuál de estos modelos tiene un mejor comportamiento tanto en la recuperación de información como una vez realizado el proceso de retroalimentación.

Referencias

[1] Dik L. Lee, Huei Chuang, Kent Seamons. Document Ranking and the Vector-Space Model. 1997

[2] MacAvaney, Sean and Yates, Andrew and Feldman, Sergey and Downey, Doug and Cohan, Arman and Goharian, Nazli. Simplified Data Wrangling with ir_datasets, 2021, SIGIR.

[3] Gesare Asnath Tinega, Waweru Mwangi, Richar Rimiru. Text Mining in Digital Libraries using OKAPI BM25 Model. 2018