UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática

Universitat Politècnica de València

Optimización de un sistema de indexación y búsqueda de palabras clave en grandes colecciones de imágenes de texto manuscrito

Proyecto Final de Carrera

Grado en Ingeniería Informática

Autor: Ernesto Noya Garcia

Director: Enrique Vidal Ruiz

Co-Director: Alejandro Hector Toselli

31/08/2015

Resumen

El aumento de libros manuscritos digitalizados, y la falta de herramientas precisas para clasificarlos, ha motivado la creación de un consorcio de universidades europeas bajo el cual la universidad politécnica de Valencia desarrolló un sistema de indexación y búsqueda de palabras clave que permite al usuario consultar y analizar el contenido de distintas colecciones de libros. En éste trabajo, hemos optimizado partes del sistema mencionado para reducir el consumo de memoria, y se ha extendido la implementación para permitir usar múltiples palabras en la búsqueda, con esto logramos una mejora en la eficiencia y flexibilidad de la herramienta, aumentando su viabilidad.

Palabras clave: Reconocimiento de texto manuscrito, Búsqueda de palabras clave, Indexación y búsqueda, Recuperación de imágenes de documentos

Tabla de contenidos

Introducción

En la actualidad, gracias al avance de internet y de las tecnologías de digitalización de libros, grandes colecciones de imágenes de libros manuscritos están siendo publicadas en internet por librerías de todas partes del mundo. Muchas de éstas colecciones contienen información histórica útil pero la cantidad de documentos es tal que la única forma de aprovechar dicha información es utilizando métodos automáticos de reconocimiento y búsqueda.

Desafortunadamente, tanto las técnicas de reconocimiento de caracteres (OCR) como las técnicas modernas de reconocimiento de texto manuscrito (HTR) no resultan adecuadas para este tipo de imágenes, las primeras por no poder separar de una manera confiable los caracteres en cursiva, y las segundas por no obtener suficiente precisión.

Con el fin de trabajar en este problema, grupos de investigación de varias universidades, entre los que se encuentra el Pattern Recognition and Human Language Technology (PRHLT) de la UPV, han creado una iniciativa conjunta denominada TranScritorium con el objetivo de desarrollar soluciones innovadoras y eficientes para la indexación, búsqueda y transcripción completa de documentos históricos manuscritos.

Como parte de esta iniciativa, han desarrollado un sistema para la indexación y búsqueda de palabras clave utilizando un modelo de precisión y exhaustividad. Este sistema, que explicamos con mas detalle en el siguiente apartado, permite al usuario realizar búsquedas en las distintas colecciones, calibrado el umbral de confianza mínima que se quiere tener en la palabra para obtener el balance deseado entre las dos métricas.

La implementación actual ha logrado conseguir buenos resultados y ha demostrado ser una herramienta de utilidad, brindando al usuario una rápida manera de realizar consultas en distintos niveles de profundidad y poder comparar fácilmente libros de una colección, capítulos de un libro, paginas de un capitulo o lineas de una página. Sin embargo ésta versión inicial también mostró 2 deficiencias, por un lado, el consumo de memoria aumenta considerablemente por cada libro que se agrega, generando un cuello de botella, y por el otro, solo se permite especificar una única palabra por búsqueda, lo que restringe las posibilidades de búsqueda.

El objetivo de este trabajo de fin de grado es solucionar dichas deficiencias, mejorando la escalabilidad del sistema, la experiencia final del usuario y, en ultima instancia la viabilidad del producto.

Para describir el proceso que se ha seguido, se dividirá esta memoria en 5 apartados, un primer apartado que introducirá el trabajo realizado por el PRHLT y las técnicas utilizadas, un segundo que explicará el marco estadístico usado en el reconocimiento, un tercero para las optimizaciones realizadas como solución al problema del consumo de recursos, un cuarto que presentara las modificaciones hechas para permitir la búsqueda de múltiples palabras en las colecciones y un último apartado para las pruebas realizadas sobre las búsqueda compuestas.

1. Descripción del sistema inicial

Como se ha mencionado, el sistema que queremos optimizar en este trabajo esta basado en el modelo de precisión y exhaustividad, esto implica que buscaremos maximizar las siguientes métricas:

Precisión: Evitar devolver resultados que no corresponden con la consulta realizada.

Exhaustividad: Obtener tantos documentos relevantes como sea posible.

Eficiencia: En el consumo de recursos y tiempo de consulta.

Flexibilidad: La capacidad de permitir diversas búsquedas.

Además, para obtener buenos resultados en dichas métricas, el sistema fue diseñado siguiendo técnicas que necesitaremos conocer para avanzar el trabajo propuesto, por lo que presentamos a continuación una breve descripción de las más relevantes:

1.1 Búsqueda de palabras clave

La búsqueda de palabras clave, aunque aplicada usualmente al reconocimiento del habla, mas recientemente a sido usada también en el reconocimiento de imágenes manuscritas. Ésta estrategia consiste en encontrar posiciones dentro de los documento donde es probable que aparezca una determinada palabra con una confianza mayor a un umbral definido, siendo posible calcular la confianza en las palabras utilizando para ello un marco estadístico práctico que explicaremos en el siguiente apartado. El objetivo final con este enfoque sera intentar obtener el mayor índice de detección de palabras con el menor número de falsos positivos, es decir, maximizar tanto la precisión como la exhaustividad de las búsquedas.

1.2 Indexación de documentos

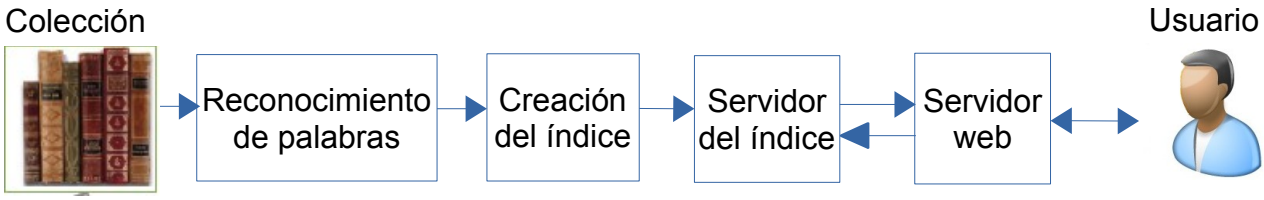
Dada la importancia de poder realizar consultas sobre las colecciones de una manera rápida, se utiliza un modelo de búsqueda basado en la indexación de documentos, donde creamos una estructura de datos que guarda la confianza de las palabras en distintos niveles y que sirve para obtener una rápida respuesta a las consultas.

1.3 Consultas como cadenas de caracteres

Para facilitar el uso por parte del usuario final, las consultas se realizarán como cadenas de caracteres, donde se entrara una cadena de texto arbitraria y el resultado serán todas las posiciones dentro de la colección de imágenes que contengan imágenes de dicha palabra.

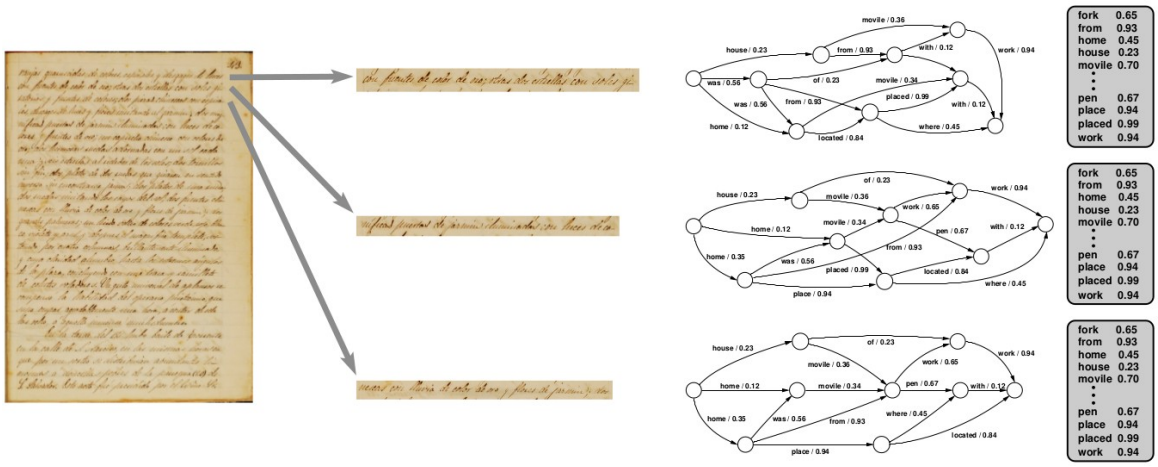
1.4 Arquitectura modular

Buscando separar los distintos módulos y automatizar gran parte del proceso, el sistema fue implementado como una tubería compuesta de las siguientes etapas:

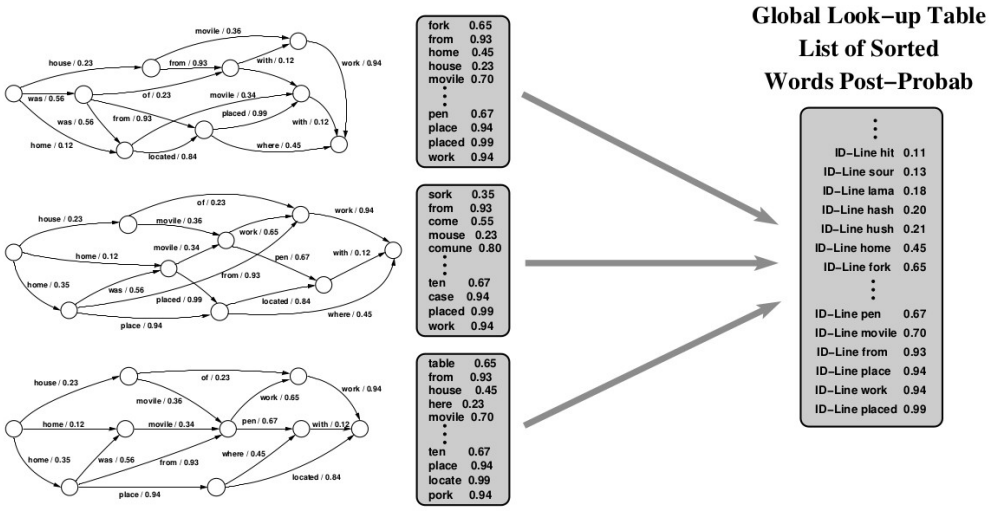


1.4.1 Reconocimiento

Como hemos visto, es necesario utilizar un marco estadístico, como la probabilidad posterior devuelta por cualquier clasificador de palabras aisladas, para calcular los valores de confianza. En ésta etapa, utilizando técnicas de segmentación y clasificación que detallaremos más adelante, se recorren las distintas páginas, obteniendo las palabras presentes en la imagen con sus respectivos valores de confianza

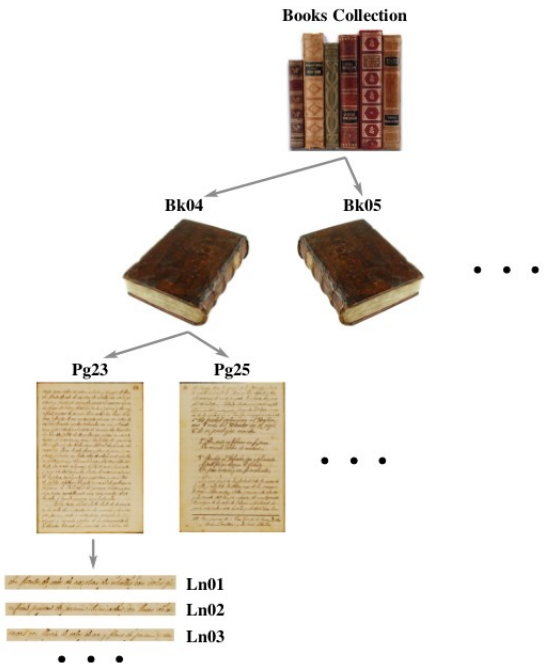


Al final del reconocimiento, el programa genera una tabla que lista hipótesis de las palabras de cada linea con un valor de confianza sobre cada una.



1.4.2 Creación del índice

Para alcanzar tanto la rapidez como la flexibilidad deseada en las búsquedas, una vez generada la lista de hipótesis, a continuación se utiliza un programa para crear y mantener actualizada una estructura de índice jerárquico que consta de 4 niveles: lineas, páginas, capítulos y libros.



En este paso es necesario definir adecuadamente las confianzas de las palabras en los distintos niveles, normalizadas y homogéneas a lo largo de la jerarquía. Para esto se agregan todas las palabras nuevas en el nivel de linea del índice y se recorren todos los registros actualizados en los niveles superiores, recalculando la confianza como la máxima de todos sus hijos. Al final de todo el proceso, el índice obtenido se guarda en un archivo de disco para ser utilizado por el siguiente programa.

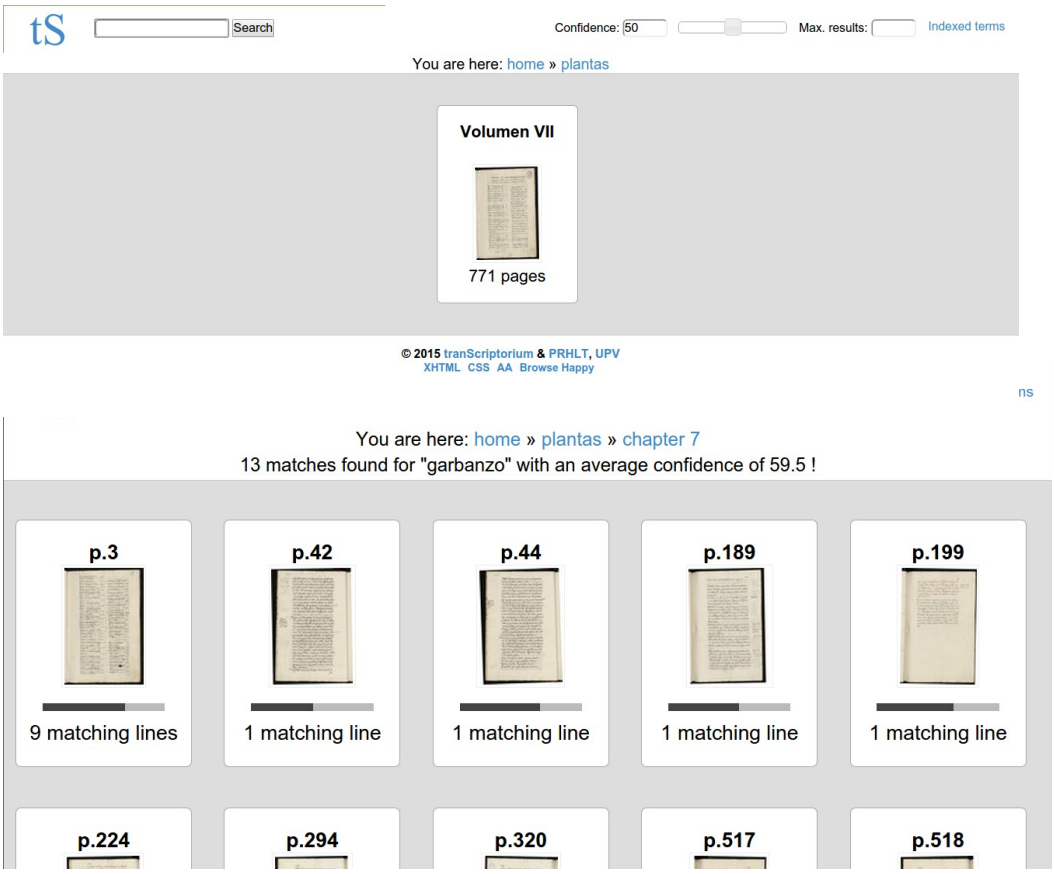
1.4.3 Servidor del índice

Siguiendo el modelo de indexación y búsqueda, que requiere hacer ambos procesos de forma asíncrona, la siguiente etapa carga de disco el índice mas actualizado y permite buscar una palabra en el mismo a través de un servidor HTTP, devolviendo como respuesta un objeto json con las posiciones de la palabra en los distintos niveles.

1.4.4 Servidor Web

Por último, como capa de interfaz al usuario se ha implementado un sitio web que ofrece una herramienta simple y abierta para buscar en las colecciones, formateando la consulta que se envía al servidor del índice jerárquico y la respuesta que se obtiene de éste.

Con el fin de mantener la interacción con el sistema lo mas sencilla y fluida posible, la página esta diseñada para poder navegar rápidamente entre los distintos niveles, utilizando una caja de texto para la búsqueda, una para el número máximo de resultados, una barra deslizante para el umbral de confianza, e imágenes de las páginas para mostrar los resultados.



2. Marco estadístico para la búsqueda de palabras clave

En el apartado anterior hemos visto que es necesario definir un marco estadístico para hacer el reconocimiento de las palabras. Ahora, para entender el marco que utilizaremos en el sistema, veremos distintos modelos estudiados para el reconocimiento de texto manuscrito.

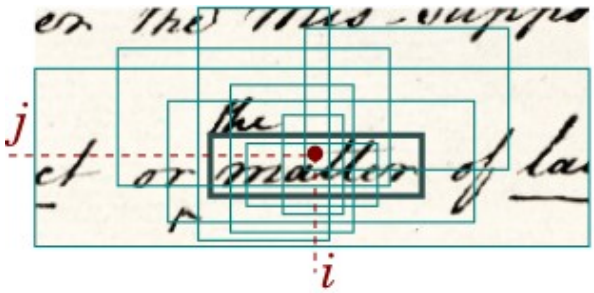
2.1 Probabilidades a nivel de píxel, "Posteriograma 2D".

$$P(v \mid X, i, j),\ 1 \le i \le I,\ 1 \le j \le J,\ v \in V$$

Donde X es una imagen de texto de tamaño IxJ, V es un vocabulario, e (i,j) un píxel de X. P(v | X,i,j) denota la probabilidad de que una palabra v esté escrita en una subimagen de X que incluye el píxel (i,j). Podemos calcular ésta probabilidad usando marginalización:

$$P(v \mid X,i,j) = \sum_{B} P(v, B|X, i, j) = \cdots \approx {}^{1}\!/_{K(i,j)} \sum_{B \in B(i,j)} P(v|X, B)$$

Donde B(i,j) es un conjunto de toda las K(i,j) regiones o subimagenes de X con forma y tamaño razonable que contienen el píxel (i,j).



Ejemplo: Algunas posibilidades de B ∈ B(i,j) para v = "matter", el valor máximo de P(v | X,B) será devuelto por la caja región con el borde remarcado.

P(v| X,B) es la probabilidad posterior (implícita o explicita) usada por cualquier clasificador de imágenes de palabras aisladas, en otras palabras, cualquier sistema capaz de solucionar el siguiente problema de clasificación para una subimagen presegmentada de una palabra de X limitada por B; claramente, mientras mejor sea el clasificador, mejor será el posteriograma estimado.

$$\hat{v} = \underset{v \in V}{argmax}\, P(v|X, B)$$

Por lo tanto el posteriorgrama 2D:

$$P(v \mid X,i,j) \approx \frac{1}{K(i,j)} \sum_{B \in \beta(i,j)} P(v|X,B) \quad , 1 \le i \le I, 1 \le j \le J, v \in V$$

puede esencialmente ser calculado con clasificación de palabras aisladas. Aunque la obtención directa de un posteriograma de ésta manera requiere una cantidad formidable de computación, puede ser calculada eficientemente utilizando combinaciones inteligentes de submuestreo de (i,j) y elecciones de B(i,j) como veremos más adelante.

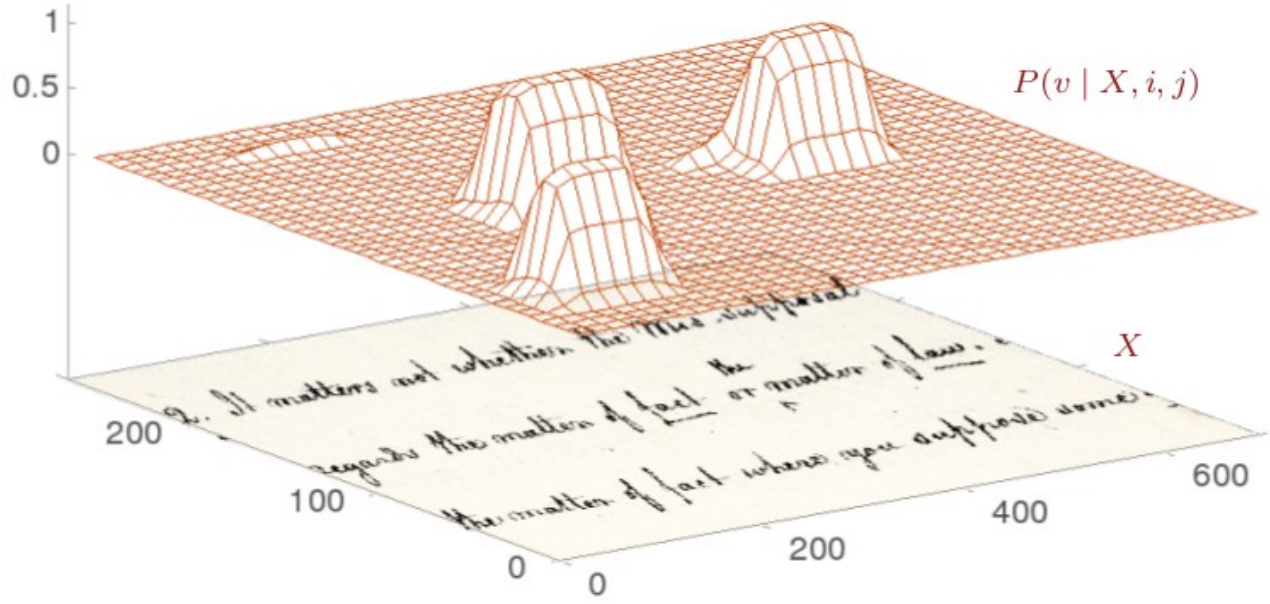


imagen de un 2D posteriograma a nivel de pixel, P (v | X, i, j), para una imagen X y la palabra v = "matter"

Una vez obtenidos, podemos utilizar éstos posteriogramas directamente para la búsqueda de palabras clave, dado un umbral $\tau \in [0, 1]$, que puede modificarse para conseguir compensaciones adecuadas de precisión y exhaustividad, una palabra $v \in V$ será observada en todas las posiciones donde $P(v \mid X,i,j) > \tau$.

Sin embargo, debido a la indexación, necesitamos la probabilidad de que una palabra v este escrita en una sección previamente especificada de la imagen, como la página, columna o linea. Para obtenerla, dada una región especifica W de la imagen X, la probabilidad W posterior, P(W | X,v), denotará la probabilidad de que v este escrita en algún lugar de X y será calculada como:

$$P(W \mid X,v) = \cdots \approx \max_{i,j} P(v|X,i,j)$$

Entonces, P(W | X,v) es el valor de confianza de una palabra propuesto en cualquier región de imagen determinada. Para simplificar la notación se ha denominado S(v, X).

2.2 Región mínima adecuada: búsqueda de palabras clave a nivel de linea.

Entre las distintas regiones de las imágenes de texto manuscrito, las lineas son útiles para la indexación y búsqueda, ya que nos permiten computar eficientemente los posteriogramas realizando un submuestreo vertical y escogiendo inteligentemente B(i,j).

Submuestreo vertical: En general, consiste en adivinar una altura de linea debida y posteriormente correr una ventana vertical de esa altura que posea alguna superposición.

Escoger B(i,j): para regiones de linea, podemos definir los bloques necesarios para calcular un posteriograma por segmentación horizontal.

En muchos casos, las lineas de texto son bastante regulares y técnicas estándar de segmentación de linea pueden obtener buenos resultados, reduciendo el tiempo de computación y en algunas ocasiones llevando a un incremento en la precisión; es posible también incrementar la robustez por medio de una sobre segmentación.

2.3 Probabilidades a nivel de linea, “Posteriograma 1D”.

Como en el caso general de 2D, podemos calcular los posteriogramas a nivel de linea por marginalización usando las probabilidades posteriores asociadas a cualquier clasificador de palabras aisladas. Sin embargo, los posteriogramas a nivel de linea pueden calcularse más eficientemente usado los llamados grafos de palabras, obtenidos como producto de aplicar el algoritmo de Viterbi, o de decodificación por paso de token, a imágenes a nivel de linea. Éstos además, proveen dos importantes ventajas adicionales:

- Utilizamos modelos morfológicos (HMM) de caracteres y Modelos N-grama de lenguajes para proveer probabilidades de clasificación de palabras, P(v|X,B), precisas y contextuales.
- Los grafos de palabras nos proveen de varias alternativas horizontales a la segmentación a nivel de palabra, que definen B(i, j) de forma directa y adecuada.

Los valores de confianza a nivel de linea, S(v, X), son directamente computados desde los correspondientes posteriogramas, S(v, x) = $\max_i$ P(v | i,x). Éstos pueden a su vez ser fácilmente combinados para obtener valores de confianza a nivel de página y superior.

Computar el posteriograma 1D:

Siendo *x* una región de la imagen a nivel de linea, representada como una secuencia de vectores de características, $x=\vec{x}_1,\vec{x}_{2,\ldots},\vec{x}_n$; como en el caso general de 2D, las probabilidades posteriores, P(v | i, x), pueden ser calculadas por marginalización:

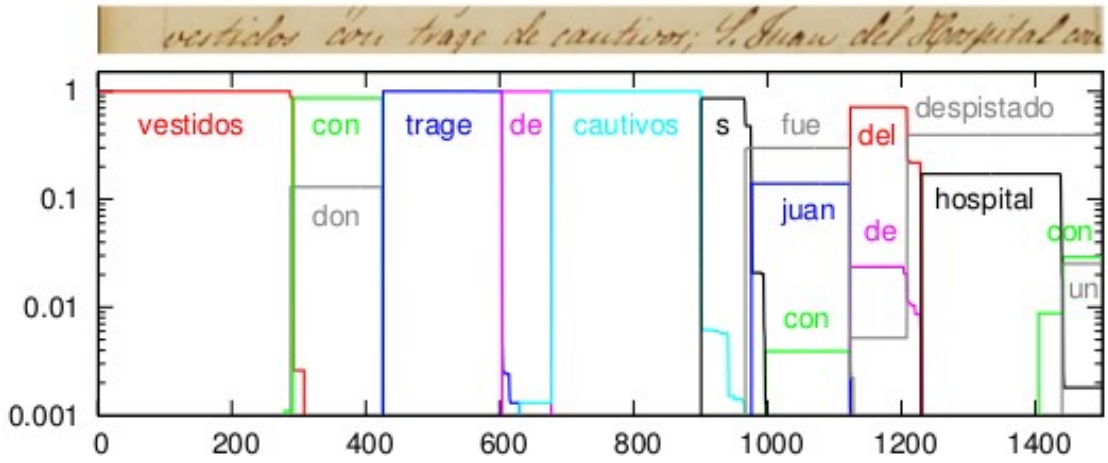
$$P(v|i,x)=\sum_{k,l} P(v,k,l|i,x)\approx \sum_{[k,l]\in\beta(i)} P(k,l|x)\cdot P(v|k,l,x)$$

donde B(i) es el conjunto (que obtenemos del grafo de palabras de *x*) de intervalos de tamaño razonable [k,l] que incluyen el frame i. Además, El grafo de palabras nos permite obtener:

- P(v | k,l,x) Probabilidad posterior de *x* para el segmento de linea x_k^l
- P(k,l | x) Probabilidad de segmento.

Alternativamente, P(v | k,l,x) podría ser obtenida por cualquier sistema capaz de reconocer imágenes presegmentadas de palabras. En éste caso, asumiendo que P(k,l | x) es uniforme para todos los K(i) intervalos de β(i):

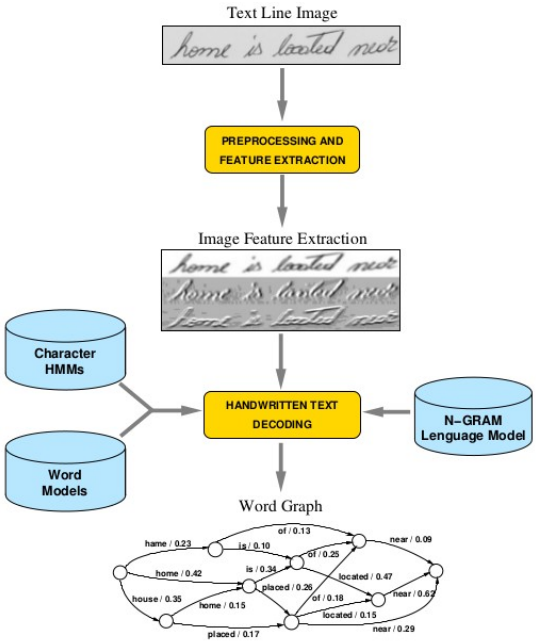
$$P(v|i,x)\approx {}^{1}\!/_{K(i)}\sum_{k,l\in\beta(i)} P(v|k,l,x)$$



Ejemplo: Para una imagen a nivel de linea dada, x, la probabilidad posterior P(v | x,i) de unas pocas palabras v son mostradas en función de la posición horizontal i.

En definitiva, el objetivo sera determinar si una palabra clave es suficientemente probable de aparecer en alguna región de linea (sin importar cuantas ocurrencias de la palabra puedan aparecer en la región o donde estén exactamente). Para ello, medimos la confianza de una palabra clave en posiciones especificas de una imagen de linea por medio del posteriograma 1D, P(v | i, x), definido como la probabilidad de que la palabra v este presente en el frame i (posición horizontal) de la secuencia de vectores x (imagen de linea). Finalmente, la búsqueda de palabras clave es llevada a cabo computando S(v, x) para cada región de linea *x* y marcando candidatas aquellas regiones donde S(v, x) > τ .

2.4 Generación del grafo de palabras:



Modulo de preproceso: realiza la normalización de los atributos del texto manuscrito.

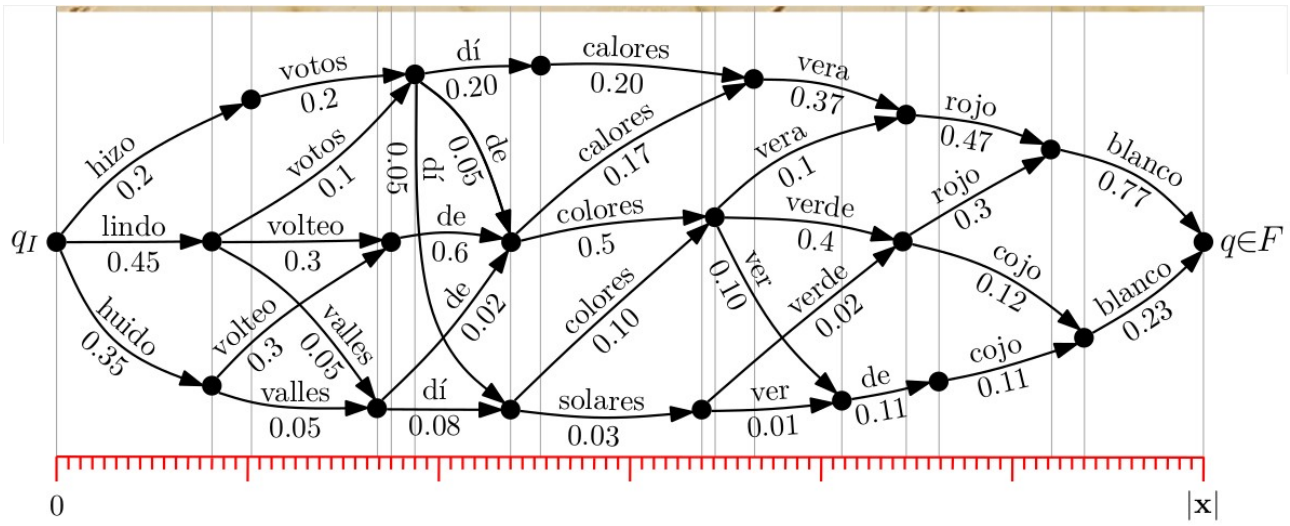
Modulo de extracción de características: transforma cada imagen de linea en una secuencia de vectores de características

$$x = \vec{x}_1, \vec{x}_{2,\ldots}, \vec{x}_n \quad , \quad \vec{x}_i \in \Re^D$$

Modulo de decodificación: dado x, encontrar las n secuencias de palabras mas probables:

$$\left\{\hat{w}_{1,\ldots}, \hat{w}_L\right\} = \underset{w}{nbest}\, p(x|w) \cdot P(w)$$

Podemos convenientemente utilizar un grafo de palabras para arreglar grandes conjuntos de n-mejores hipótesis:



En el grafo, cada nodo q esta asociado con una posición horizontal t(q) ∈ (0, |x|] y para cada limite (q', q); v = ω(q', q) es una palabra y p(q', q) es un valor de limite, que normalizado se denotará φ(q', q).

Normalización del grafo y probabilidades posteriores de los limites:

Podes realizar el cálculo de las probabilidades posteriores de los limites, φ(q', q), con una versión para grafos de palabras del conocido algoritmo de avance-retroceso:

$$\varphi(q',q)=\alpha(q')\cdot p(q',q)\cdot\beta(q)\Big/\beta(q_I)$$

donde:

$$\alpha(q)=\begin{cases}1 & \text{si } q=q_I\\ \sum_{q':(q',q)\in E}\alpha(q)\cdot p(q,q) & \text{en otro caso}\end{cases}$$

$$\beta(q)=\begin{cases}1 & \text{si } q\in F\\ \sum_{q'':(q,q'')\in E} p(q,q'')\cdot\beta(q'') & \text{en otro caso}\end{cases}$$

y se puede demostrar la siguiente afirmación:

$$\sum_{\varphi\in\Phi}\prod_{(q',q)\in\varphi} p(q',q)=\sum_{q\in F}\alpha(q)=\beta(q_I)$$

siendo Φ todos los posibles caminos del grafo de palabras desde un estado q_I hasta cualquier estado $q \in F$.

Costes de generación y normalización del grafo:

El algoritmo de Viterbi tiene una complejidad computacional O(n) y puede hacerse independiente del tamaño del vocabulario y modelo utilizando búsqueda de haz y poda. Cuando se incluye la generación de un grafo de palabras de longitud n = |x|, el coste computacional del proceso de decodificación completo se observa en O(Γ · n), donde Γ es una constante que crece rápidamente con el tamaño del grafo.

Por otro lado, la complejidad computacional del proceso de normalización del grado es:

$$O(n \cdot m \cdot B) = O(|Q| \cdot B) = O(|E|)$$

Donde m = |Q| / n es el promedio de estados por frame y B = |E| / |Q| es el promedio del factor de ramificación del grafo de palabras. Éste coste de normalización es despreciable con respecto al coste de generara el grafo mismo (sobre 1%).

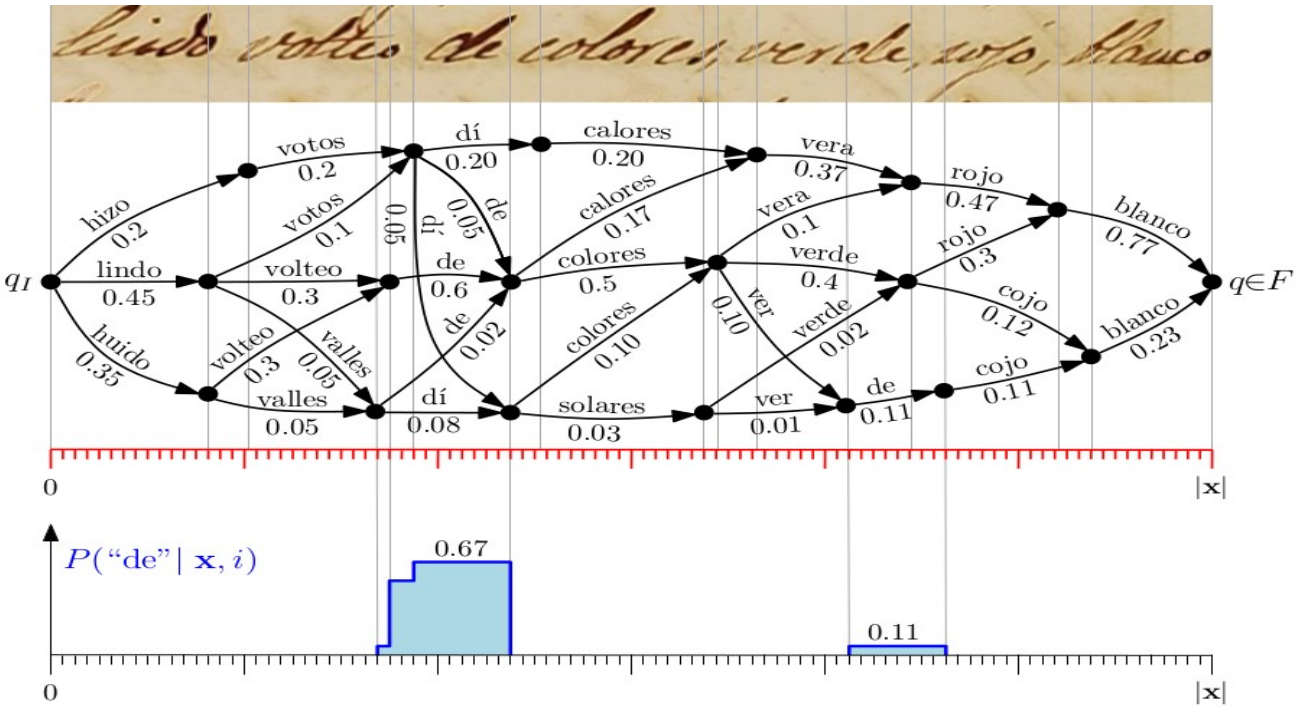
Calculando el posteriograma a nivel de linea desde el grafo de palabras:

Como mencionamos, P(v | x,i) es computada por marginalización, desde la contribución de los segmentos horizontales de imagen [k,l] que contienen la posición horizontal i:

$$P(v|i,x) \approx \sum_{[k,l] \in B(i)} P(k,l|x) \cdot P(v|k,l,x)$$

Ahora, con el conjunto de posibles segmentos B(i), junto con las correspondientes probabilidades de segmentación y clasificación, P (k,l | x) y P (v | k,l,x), provistos por el grafo de palabras, se llega finalmente a:

$$P(v|x,i) = \cdots \approx \sum_{(q',q):\, v=w(q',q),\, t(q)<i\le t(q)} \varphi(q',q)$$



Ejemplo: P(“de”|X,i) dado el grafo de palabras de la imagen de linea X(arriba) en función de la posición horizontal.

El posteriograma P(v|i,x), puede calcularse visitando secuencialmente los limites del grafo de palabras y actualizando para cada limite (q' , q), los valores de P(v | i,x) para v = ω(q' , q) y para todas las i comprendidas entre t(q') y t(q). El computo del máximo para obtener S(v, x) puede ser llevada a cabo trivialmente durante el procedimiento anterior.

El coste computacional final es proporcional al número de limites en el grafo de palabras y a la longitud media de los segmentos de *x* definidos por los limites.

3. Optimización del consumo de recursos

La primera métrica que buscamos optimizar en este trabajo es la eficiencia del sistema debido a que, aunque la estructura de índice permite obtener una velocidad de respuesta rápida, el consumo de memoria principal que genera termina limitando seriamente el número de libros que se pueden agregar.

Como hemos señalado en el apartado anterior, solo dos etapas del sistema utilizan directamente el índice: la segunda, encargada de crearlo, y la tercera, que lo carga en memoria para responder consultas. Dado que la implementación actual de éstas etapas esta escrita en el lenguaje de programación Python, un lenguaje interpretado que realiza una gestión de memoria en tiempo de ejecución, la estrategia que hemos elegido seguir para lograr alcanzar una mejora considerable en la eficiencia es volver a implementar las mismas en el lenguaje compilado C++, que ofrece mayores optimizaciones del código y manejo de la memoria.

Así, con el fin de mantener la estructura jerárquica del índice, que permite la búsqueda en los distintos niveles, se aplicaron técnicas de abstracción, polimorfismo y herencia, junto con el uso de punteros y paso de parámetros por referencia de C++ para crear una estructura homologa que posea una mejor gestión de la memoria.

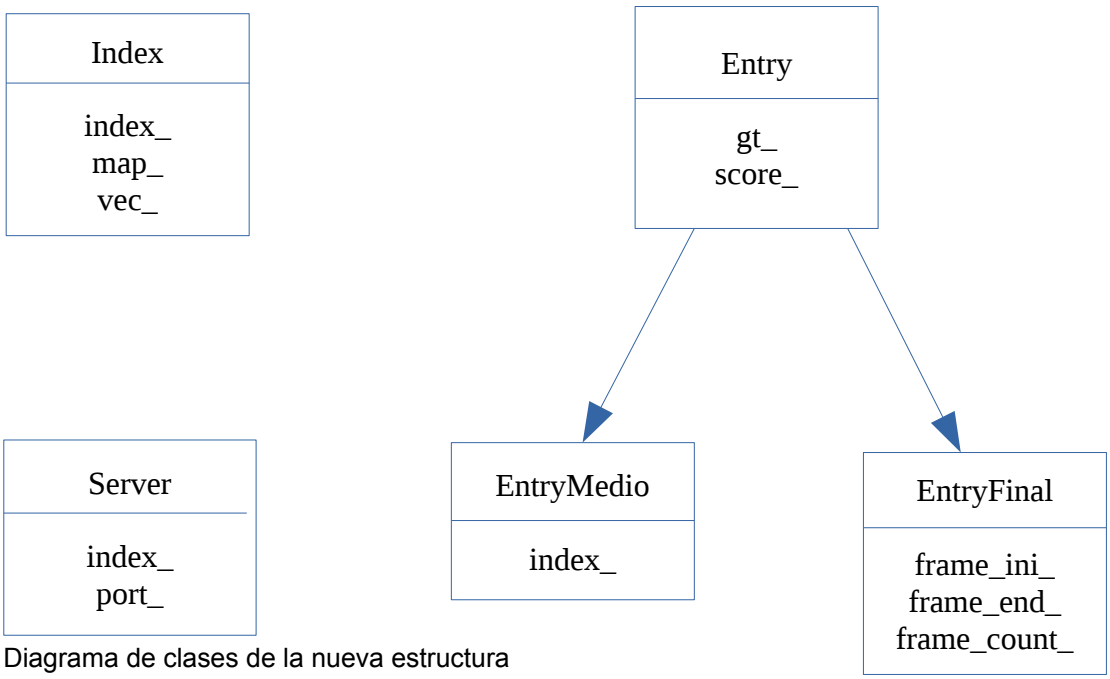


Diagrama de clases de la nueva estructura

3.1 Índice

```
class Index {
 public:
      int size_;
      bool with_scores_;

      unordered_map<string, Entry*> index_;
      unordered_map<string, int> map_;
      vector<string> vec_;

      Index();
      bool AddEntry(const list<string>& path,EntryFinal* entry);
      Entry* Search(const list<string>& path);
      string getDict(Entry * entry,int levels);
      int getID(const string& word);
      bool isIndexed(const char*);
      void computeScores();
      int size();
};
```

La nueva implementación del índice, representada por la clase “Index”, posee una tabla hash que almacena en la posición asociada a cada término una referencia a un objeto “Entry”, donde se guardan los datos de la entrada correspondiente. Además, se han agregado un vector de cadenas que almacena los identificadores internos y otra tabla que asocia cada identificador con su posición en el vector. Con estas estructuras podemos mapear las cadenas a enteros, que requieren menos memoria, y utilizarlos como identificadores en las entradas internas, guardando solo 2 copias de cada identificador.

3.2 Entradas del índice

```
class Entry {
 public:
      static char GT_UNK;
      static char GT_FALSE;
      static char GT_TRUE;

      char gt_;
      double score_;

      Entry();
      Entry(const char& gt ,const double& score);
      virtual ~Entry() {}
      virtual bool isEnd() = 0;
      char getGT() const {return gt_;}
      double getScore() const {return score_;}
};
```

La clase “Entry” crea las variables para guardar la confianza de la palabra y el etiquetado de referencia, necesarias en todos los niveles. Igualmente, al ser una clase abstracta, debe ser utilizada como uno de sus hijos, que permiten definir las diferencias entre los niveles medios y el último nivel.

```
template <typename T>

class EntryMedio :public Entry{
 public:
      typedef unordered_map<T, Entry*> index_type;
      index_type index_;

      EntryMedio();
      virtual ~EntryMedio() {}
      index_type* subindex();
      void compute_score();
      bool isEnd(){return false;}
};
```

Las entradas de los niveles medios agregan una tabla hash que asocia los identificadores del nivel inferior, transformados en enteros para utilizar menos memoria, con referencias a objetos “Entry” de ése nivel, permitiendo implementar los niveles intermedios necesarios. En este caso libros, capítulos y páginas.

```
class EntryFinal :public Entry{
 public:
      unsigned short int frame_ini_;
      unsigned short int frame_end_;
      unsigned short int frame_count_;

      EntryFinal();
      EntryFinal(const char& gt ,const double& score ,const unsigned short int&
      frame_ini ,const unsigned short int& frame_end ,const unsigned short int&
      frame_count);
      virtual ~EntryFinal() {}
      bool isEnd(){return true;}
      string toString(){
            string d =  "\"gt\":" + std::to_string(gt_) + ",\"score\":" +
            std::to_string(score_) + ",\"frame_ini\":" +
            std::to_string(frame_ini_) + ",\"frame_end\":" +
            std::to_string(frame_end_) + ",\"frame_count\":" +
            std::to_string(frame_count_);
            return d;
      }
};
```

Las entradas finales en su lugar, agregan 3 variables que sirven para almacenar las medidas de la ventana utilizada en la segmentación horizontal, permitiendo obtener la posición de la palabra a nivel de linea.

3.3 Servidor del índice

```
class Server {
private:
      Index * index_;
      unsigned short int port_;
```

```
public:
	Server(unsigned short int& port);
	int start();
	void loadIndex(Index* index);
	static int answer_to_connection (void *cls, struct MHD_Connection
		*connection,
		const char *url,
		const char *method, const char *version,
		const char *upload_data,
		size_t *upload_data_size, void **con_cls);
	static int print_out_key (void *cls, enum MHD_ValueKind kind,
		const char *key, const char *value);
};
```

Respetando la arquitectura modular, se utiliza la clase Index para implementar dos programas, uno que crea o actualiza una instancia de la clase y usando la librería "Serialization" del paquete Boost de C++ guarda la instancia en disco, y otro que utiliza la clase "Server" para cargar un índice de disco y con la librería "libmicrohttpd", arrancar un servidor HTTP en un puerto determinado para responder consultas al índice.

3.4 Tabla comparativa

Las pruebas realizadas con las colecciones actuales demuestran que gracias a la reimplementación del código y a las estrategias seguidas, hemos conseguido un descenso considerable del espacio utilizado en memoria principal, como vemos en la siguiente tabla obtenida con la colección de prueba "Plantas".

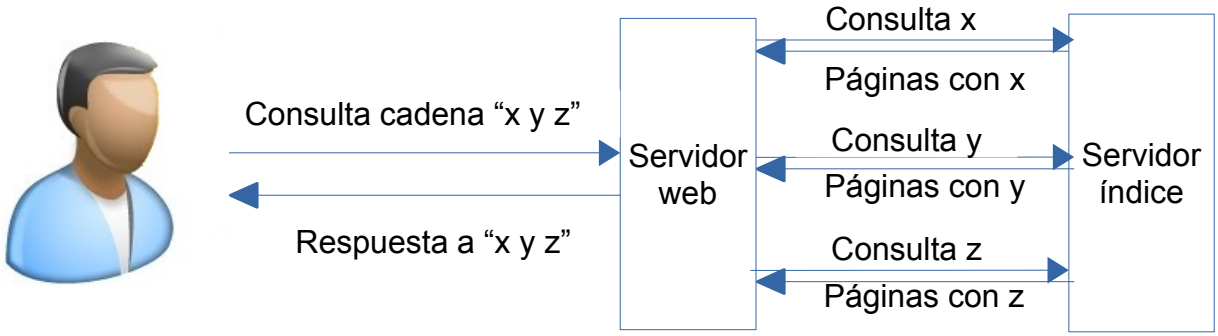
	Sistema inicial	Sistema optimizado
Tamaño del índice en disco	426 Mb	188 Mb
Tamaño del índice en RAM	8 Gb	750 Mb
Tiempo de creación	-	1min 30 s
Tiempo de carga	> 2min	40 s

Tabla 1:Comparación de la eficiencia del índice entre la versión en Python y C++

4. Mejoras a la interfaz de usuario

La segunda métrica a mejorar en el trabajo es la flexibilidad, dado que, aunque el sistema permite realizar fácil y rápidamente búsquedas de una palabra en los distintos niveles, carece de la opción de realizar búsquedas de múltiples palabras, presente en prácticamente todos los buscadores modernos.

En este apartado, y de nuevo siguiendo la arquitectura existente, la estrategia elegida fue implementar modificaciones en la interfaz al usuario que permitan realizar búsquedas conjuntas utilizando para ello peticiones al índice de cada palabra aislada y operaciones del álgebra de conjuntos.



Siguiendo dicha estrategia se han implementado 4 operadores lógicos para realizar búsquedas compuestas; 3 que permiten operar con la presencia de palabras a nivel de página, que resulta más intuitivo al poder obtener mas contexto que en una única linea, y uno que permite la búsqueda a nivel de linea, ideal para frases o expresiones especificas.

4.1 OR “||”

Ésta operación puede utilizarse con el símbolo “||” entre las palabras para buscar elementos del nivel correspondiente que contengan al menos una de las mismas con confianza mayor al umbral definido.

Para lograrlo, el sitio web consulta al servidor del índice por cada palabra y hace una unión de las distintas respuestas, utilizando como nueva aproximación de la confianza en los niveles intermedios la máxima confianza obtenida entre las distintas palabras.

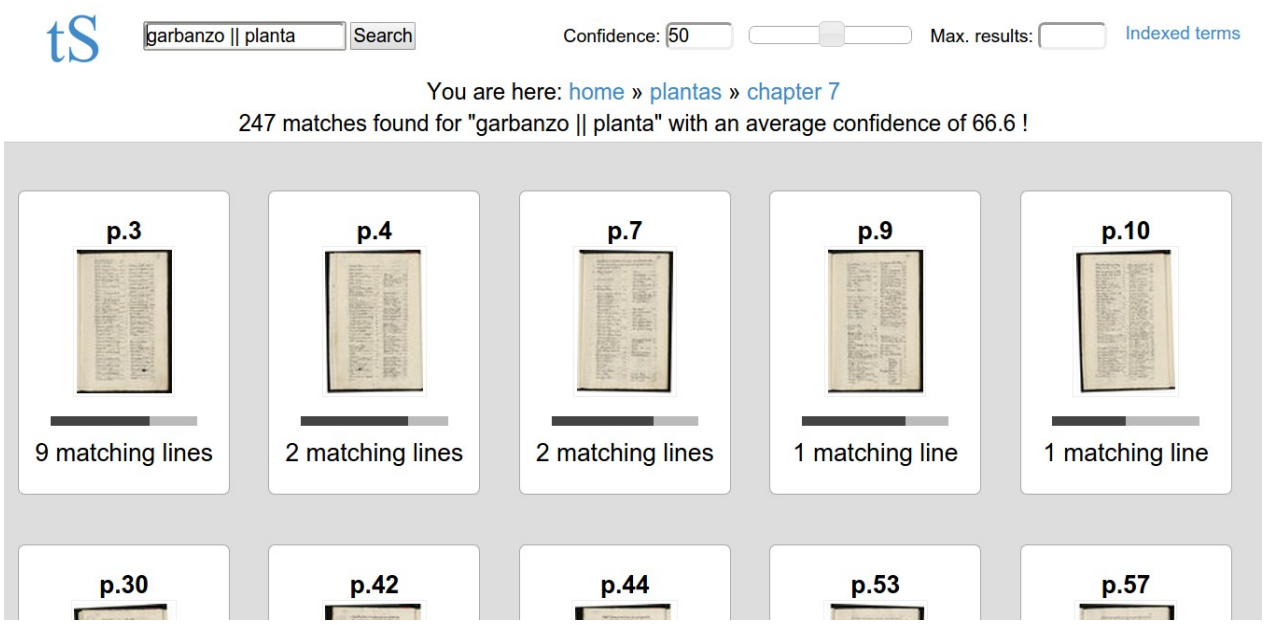


Imagen 1: resultado de la búsqueda "garbanzo || planta" a nivel de capitulo en la colección Plantas.

4.2 AND "&&"

Ésta operación puede utilizarse con el símbolo "&&" entre las palabras, o por defecto cuando no se especifica ningún operador, para buscar elementos del nivel que contengan páginas con al menos una aparición de cada palabra con confianza mayor al umbral.

Esto lo logra haciendo una interacción de los resultados de las distintas palabras y utilizando como nueva aproximación de la confianza la mínima confianza entre las palabras.

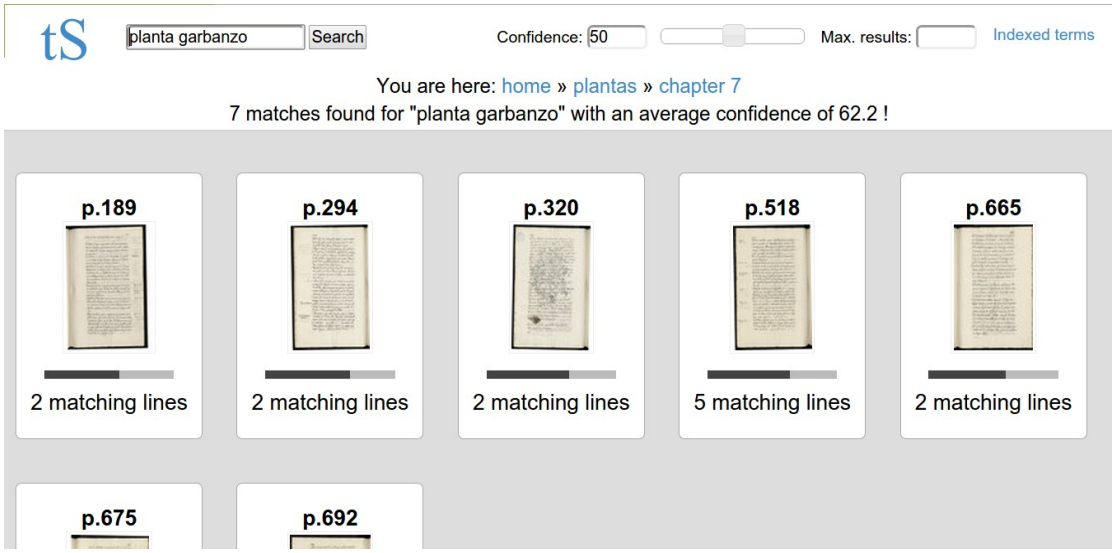


Imagen 2: resultado de la búsqueda "planta garbanzo" a nivel de capitulo

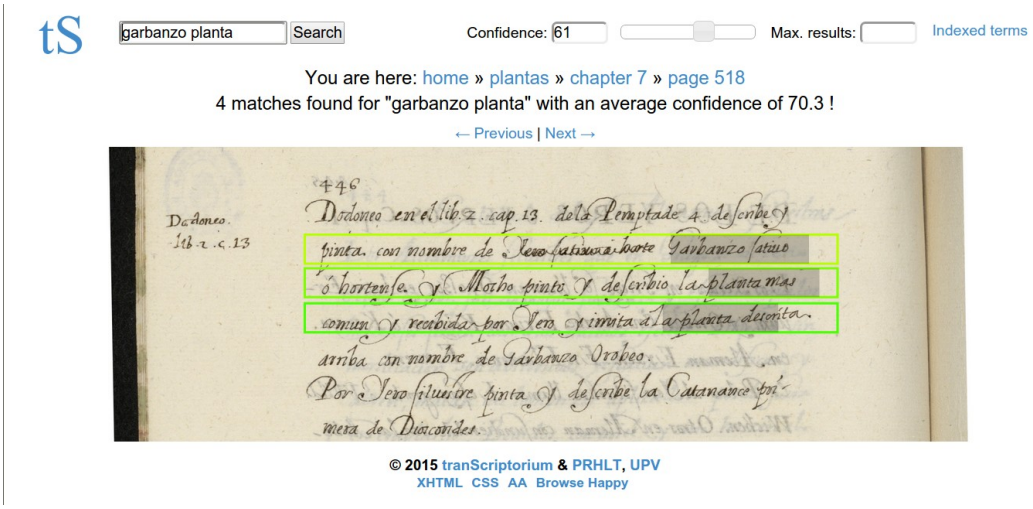


Imagen 3: resultado de la búsqueda "garbanzo planta" a nivel de página

Para alinear el uso del sistema con la intuición del usuario, el operador se ha implementado de forma que si se utiliza dentro de una página, al igual que el OR, devuelve las lineas que contengan al menos una de las palabras.

4.3 NOT "-"

Ésta operación puede utilizarse con el símbolo "-" entre las palabras para extraer de los resultados obtenidos hasta el mismo, todas las páginas que tengan una aparición de la siguiente palabra con una confianza mayor al umbral.

Consigue esto restando al conjunto de respuestas anterior, las páginas que contienen la palabra a extraer, obtenidas como respuesta del índice, y utilizando como nueva aproximación para la confianza: el mínimo entre la anterior y 1 – la confianza obtenida para las páginas, y el mínimo del nivel inferior para los niveles superiores.

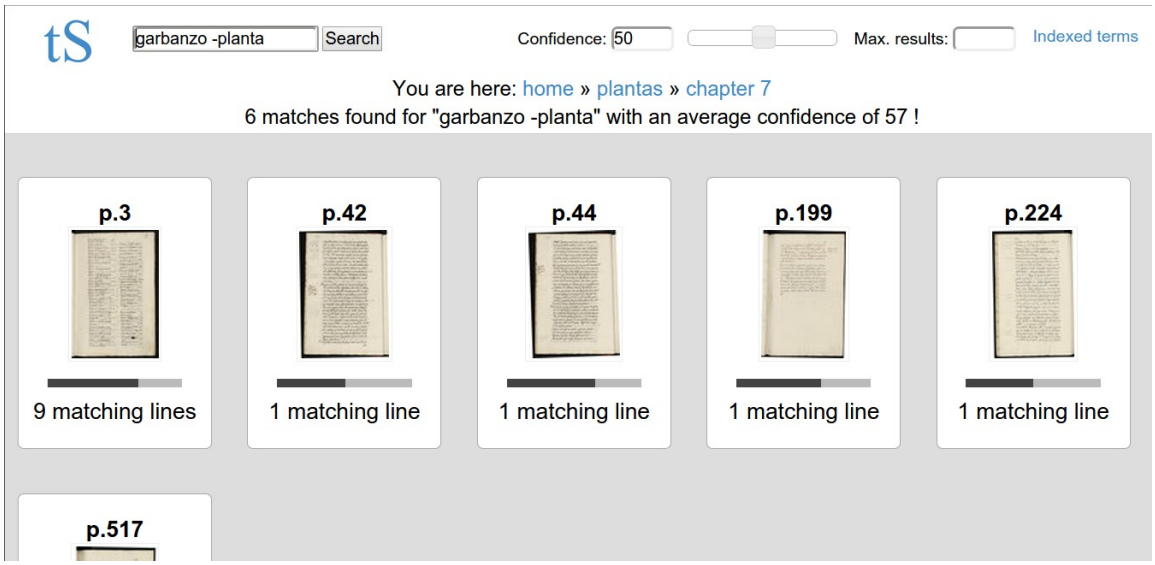


Imagen 4: resultado de la búsqueda "garbanzo -planta" a nivel de capitulo

4.4 AND Linea “//”

Ésta operación puede utilizarse con el símbolo “/” al principio y final de una cadena para buscar elementos del nivel que contengan al menos una linea con todas las palabras de la cadena con confianza mayor al umbral.

Para lograrlo realiza una intersección de las respuestas a nivel de linea, uniendo el marco horizontal que marca la posición, y utilizando como nueva aproximación a la confianza, la mínima confianza entre la palabras.

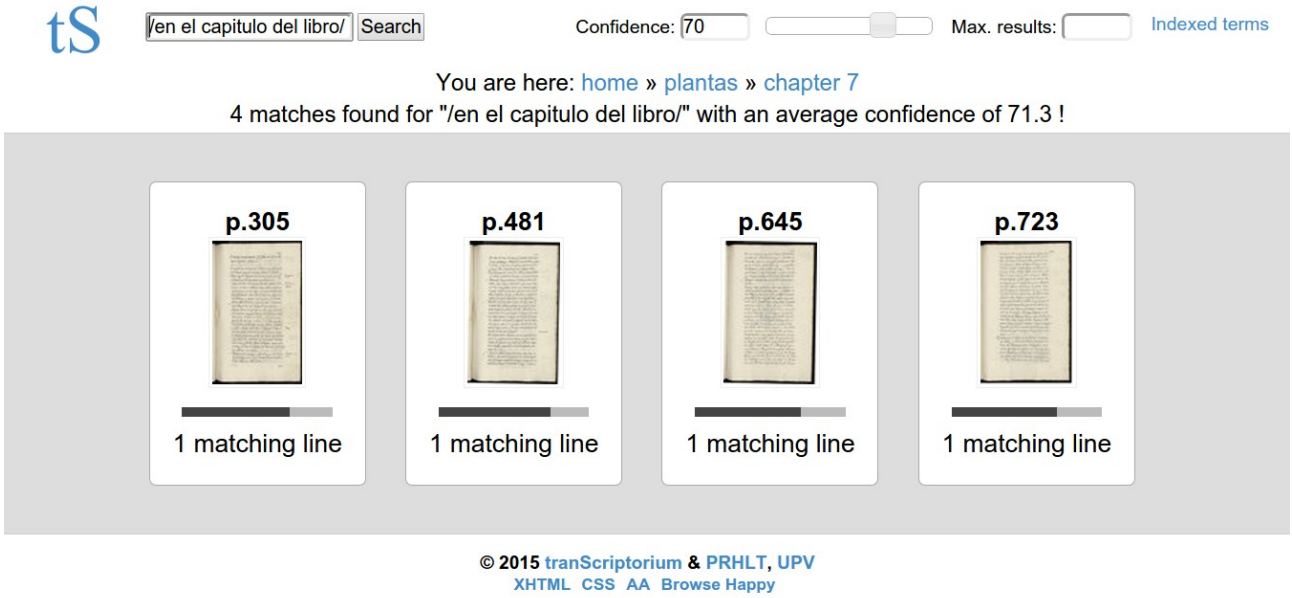


Imagen 5: resultado de la búsqueda “/en el capitulo del libro/” a nivel de capitulo

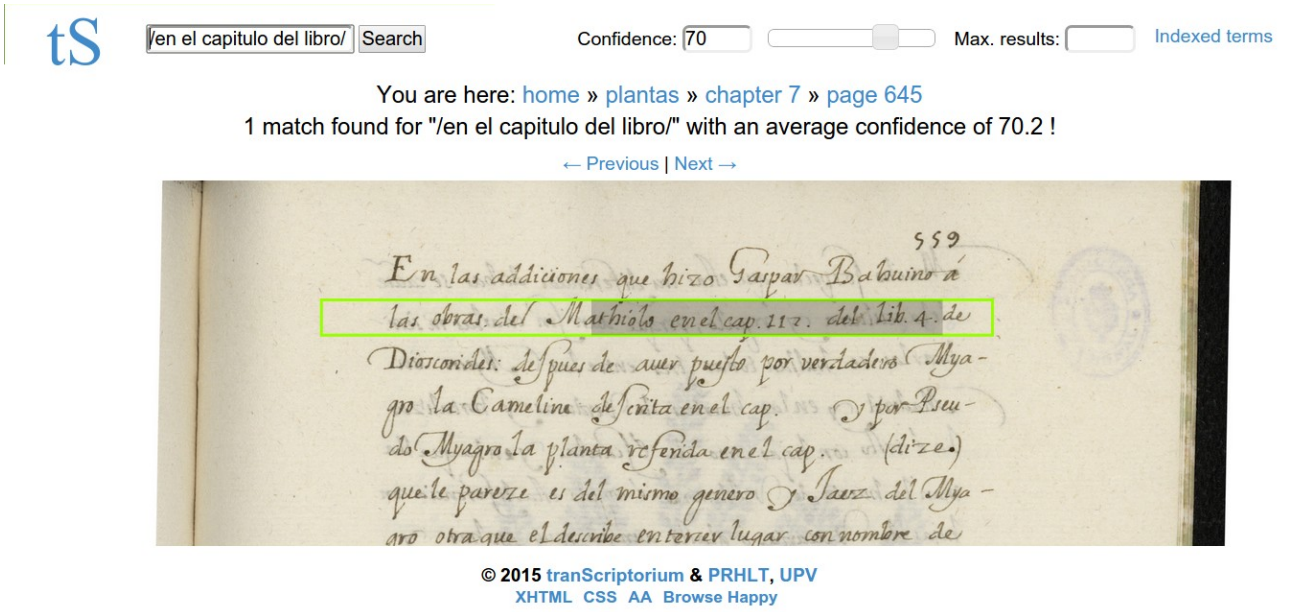


Imagen 6: resultado de la búsqueda “/en el capitulo del libro/” a nivel de página

4.5 Orden de operadores

Con el fin de definir una forma de ordenar los operadores, aumentando más aun la flexibilidad, se ha implementado también una estructura de pila que permite utilizar los símbolos paréntesis para combinar los mismos en diversas consultas de mayor complejidad.

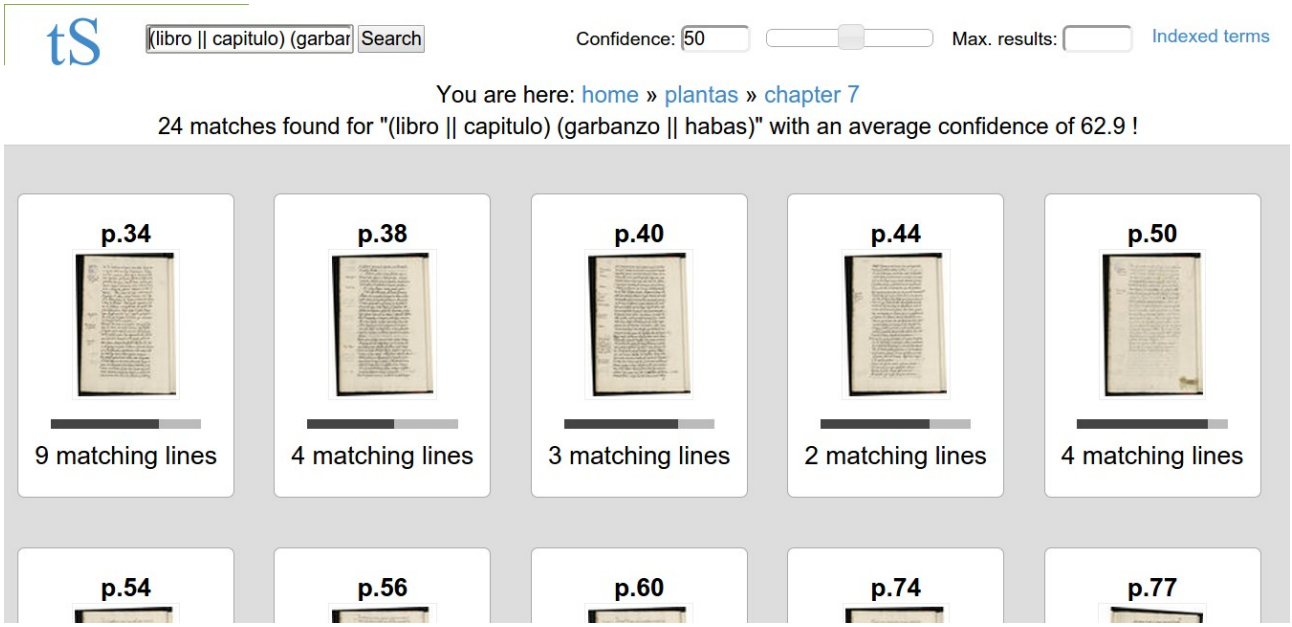


Imagen 7: resultado de la búsqueda "(libro || capitulo) (garbanzo || habas)" a nivel de capitulo

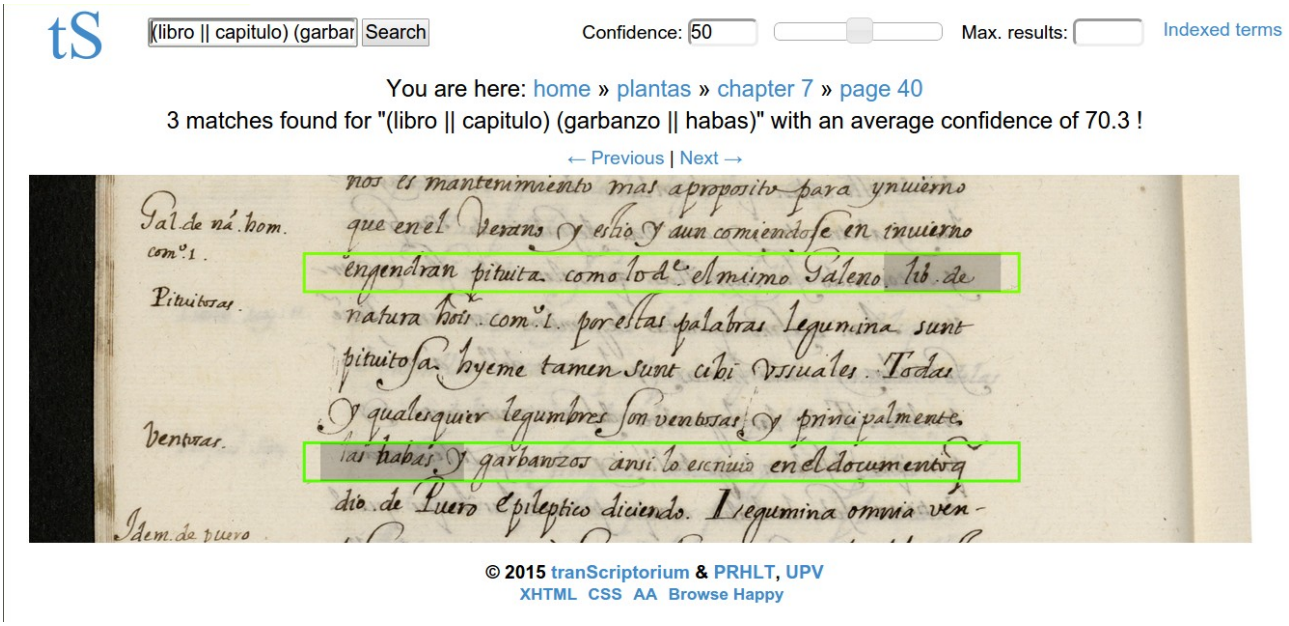


Imagen 8: resultado de la búsqueda "(libro || capitulo) (garbanzo || habas)" a nivel de página

5. Pruebas sobre las búsquedas compuestas

Con el fin de analizar el impacto de las aproximaciones usadas en los resultados obtenidos por el sistema, hemos realizado un pequeño conjunto de pruebas que nos permitirán obtener los valores de precisión y exhaustividad de las búsquedas compuestas. Él cálculo de éstas métricas del modelo, que describimos a continuación, nos ayudará a definir la veracidad de los resultados de las búsqueda.

5.1 Descripción del cálculo de precisión y exhaustividad

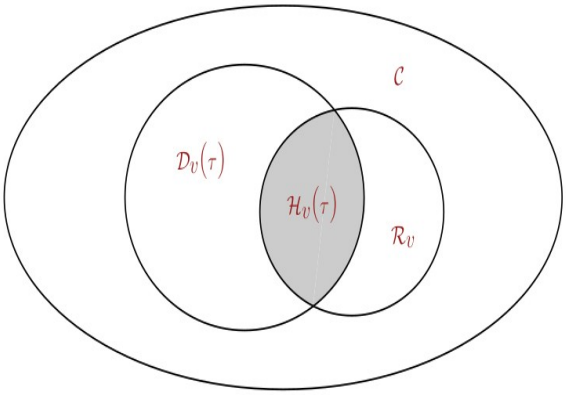
Dados*:*

C: Colección de elementos

R_v: Conjunto de los elementos relevantes para la consulta v dentro de C

$D_v(\tau)$: Conjunto de elementos detectados para la consulta v y umbral τ

$H_v(\tau)$: Conjunto de elementos relevantes detectados, $R_v \cap D_v(\tau)$, para una consulta v y umbral t



Definiremos la precisión $\pi_v(\tau)$ para una consulta determinada v y umbral $\tau \in [0, 1]$ como:

$$\pi_v(\tau) = \frac{|R_v \cap D_v(\tau)|}{|D_v(\tau)|} = \frac{h_v(\tau)}{d_v(\tau)}$$

Y definiremos la exhaustividad $\rho_v(\tau)$ para v y $\tau \in [0, 1]$ como:

$$\rho_v(\tau) = \frac{|R_v \cap D_v(\tau)|}{|R_v(\tau)|} = \frac{h_v(\tau)}{r_v(\tau)}$$

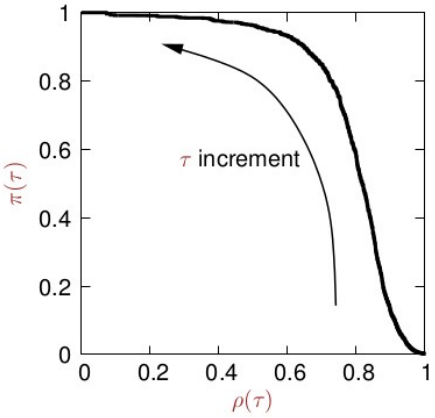
O expresadas en término de verdaderos positivos (TP), falsos positivos (FP) y falsos negativos (FN):

$$\pi_v(\tau) = \frac{TP(\tau)}{TP(\tau) + FP(\tau)}$$

$$\rho_v(\tau) = \frac{TP(\tau)}{TP(\tau) + FN(\tau)}$$

5.2 Consultas de prueba

Las siguientes pruebas las hemos realizado sobre una colección de 83 páginas del filósofo y jurista inglés Jeremy Bentham, de las cuales disponemos de suficiente información para determinar los conjuntos R_v, $D_v(\tau)$ y $H_v(\tau)$ de cada consulta v en distintos umbrales de confianza τ, y obtener gráficas de las curvas de compensación entre precisión y exhaustividad además de calcular la precisión media interpolada como el área bajo la curva.



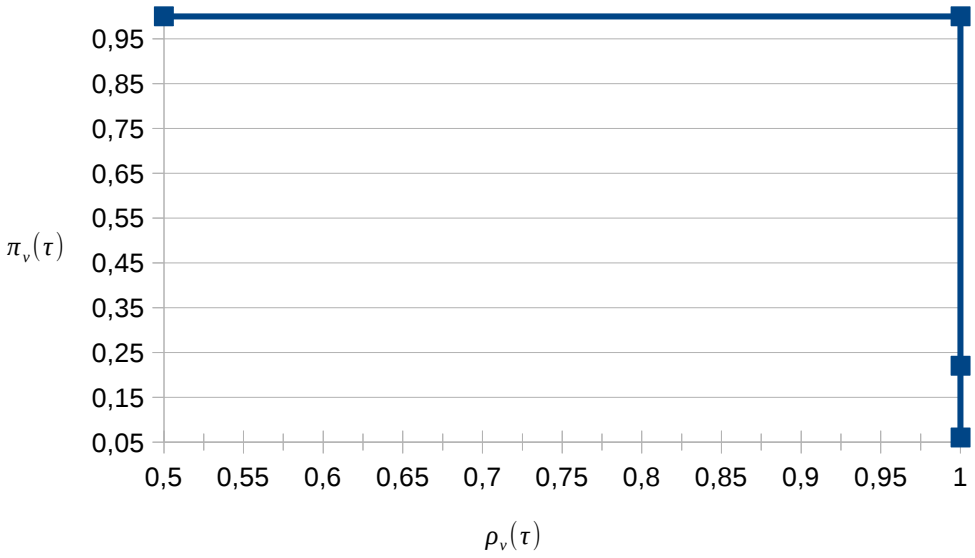
Consulta: “matter law”

Elementos relevantes *(R_v)*: 2

Calculo de $\pi_v(\tau)$ y $\rho_v(\tau)$ para *τ = 0,0.2,0.4,0.6,0.8,1:*

τ	$\lvert D_v(\tau)\rvert$	$\lvert H_v(\tau)\rvert$	$\pi_v(\tau)$	$\rho_v(\tau)$
0	33	2	0,06	1
0,2	9	2	0,22	1
0,4	2	2	1	1
0,5	2	2	1	1
0,6	1	1	1	0,5
0,8	0	0	0	0
1	0	0	0	0

Curva de precisión/exhaustividad:



Precisión media interpolada: 1

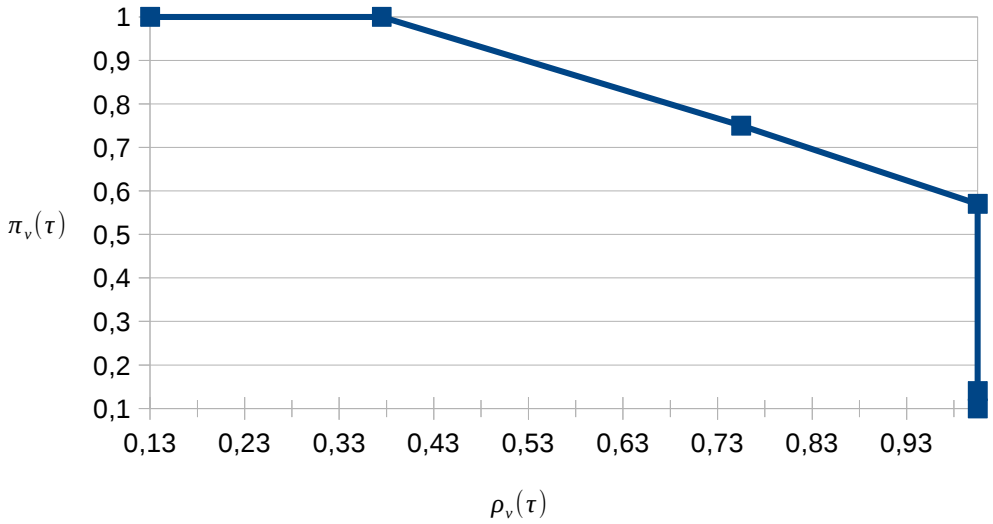
Consulta: “one time”

Elementos relevantes *(R_v)*: 8

Calculo de $\pi_v(\tau)$ y $\rho_v(\tau)$ para *τ = 0,0.2,0.4,0.6,0.8,1:*

τ	$\lvert D_v(\tau)\rvert$	$\lvert H_v(\tau)\rvert$	$\pi_v(\tau)$	$\rho_v(\tau)$
0	78	8	0,1	1
0,2	56	8	0,14	1
0,4	14	8	0,57	1
0,5	8	6	0,75	0,75
0,6	3	3	1	0,37
0,8	1	1	1	0,125
1	0	0	0	0

Curva de precisión/exhaustividad:



Precisión media interpolada: 0.864

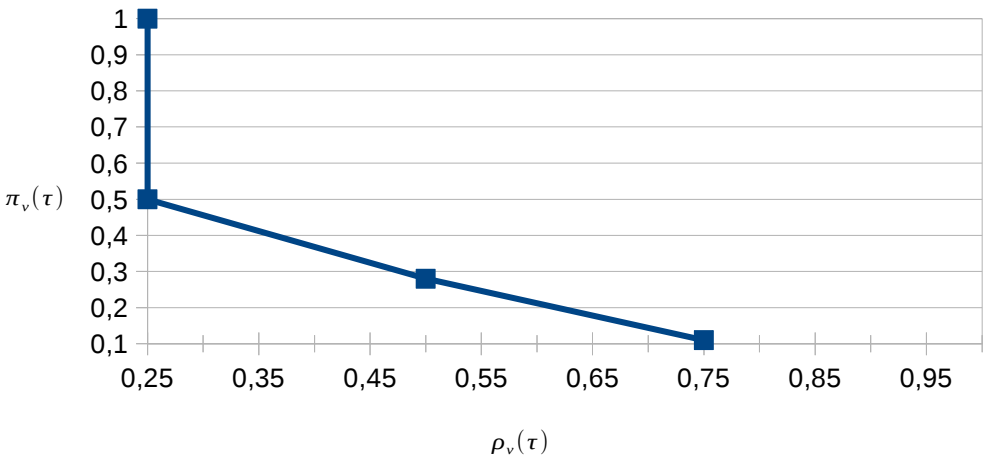
Consulta: “simple case”

Elementos relevantes *(R_v)*: 4

Calculo de $\pi_v(\tau)$ y $\rho_v(\tau)$ para *τ = 0,0.2,0.4,0.6,0.8,1:*

τ	$\lvert D_v(\tau)\rvert$	$\lvert H_v(\tau)\rvert$	$\pi_v(\tau)$	$\rho_v(\tau)$
0	26	3	0,11	0,75
0,2	7	2	0,28	0,5
0,4	2	1	0,5	0,25
0,5	1	1	1	0,25
0,6	0	0	0	0
0,8	0	0	0	0
1	0	0	0	0

Curva de precisión/exhaustividad:



Precisión media interpolada: 0.425

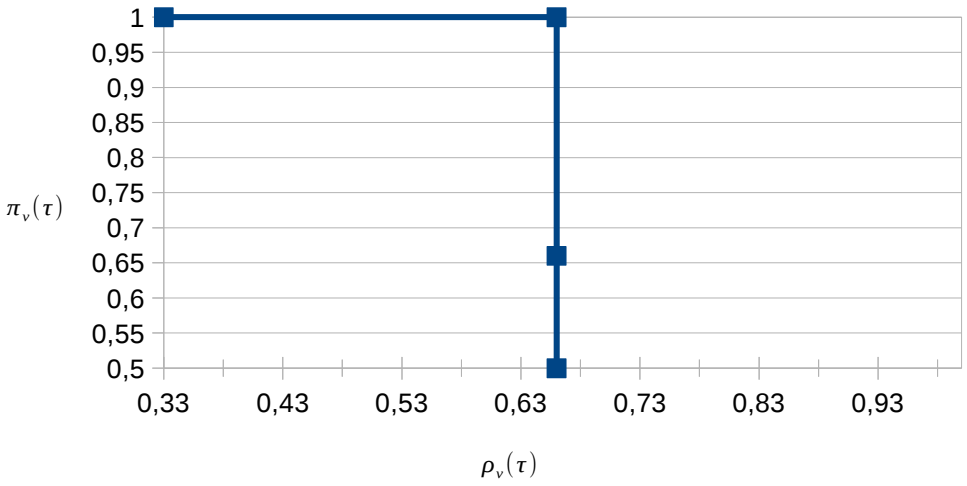
Consulta: “injuries damage”

Elementos relevantes *(R_v)*: 3

Calculo de $\pi_v(\tau)$ y $\rho_v(\tau)$ para *τ = 0,0.2,0.4,0.6,0.8,1:*

τ	$\lvert D_v(\tau)\rvert$	$\lvert H_v(\tau)\rvert$	$\pi_v(\tau)$	$\rho_v(\tau)$
0	4	2	0,5	0,66
0,2	3	2	0,66	0,66
0,4	2	2	1	0,66
0,5	1	1	1	0,33
0,6	1	1	1	0,33
0,8	1	1	1	0,33
1	0	0	0	0

Curva de precisión/exhaustividad:



Precisión media interpolada: 0.66

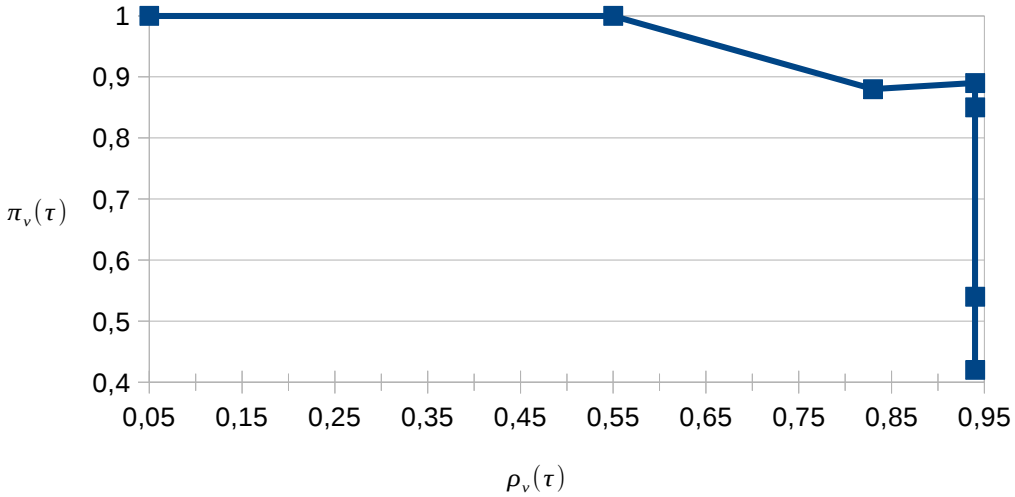
Consulta: “public || property”

Elementos relevantes *(R_v)*: 18

Calculo de $\pi_v(\tau)$ y $\rho_v(\tau)$ para *τ = 0,0.2,0.4,0.6,0.8,1:*

τ	$\lvert D_v(\tau)\rvert$	$\lvert H_v(\tau)\rvert$	$\pi_v(\tau)$	$\rho_v(\tau)$
0	40	17	0,42	0,94
0,2	31	17	0,54	0,94
0,4	20	17	0,85	0,94
0,5	19	17	0,89	0,94
0,6	17	15	0,88	0,83
0,8	10	10	1	0,55
1	1	1	1	0,05

Curva de precisión/exhaustividad:



Precisión media interpolada: 0.918

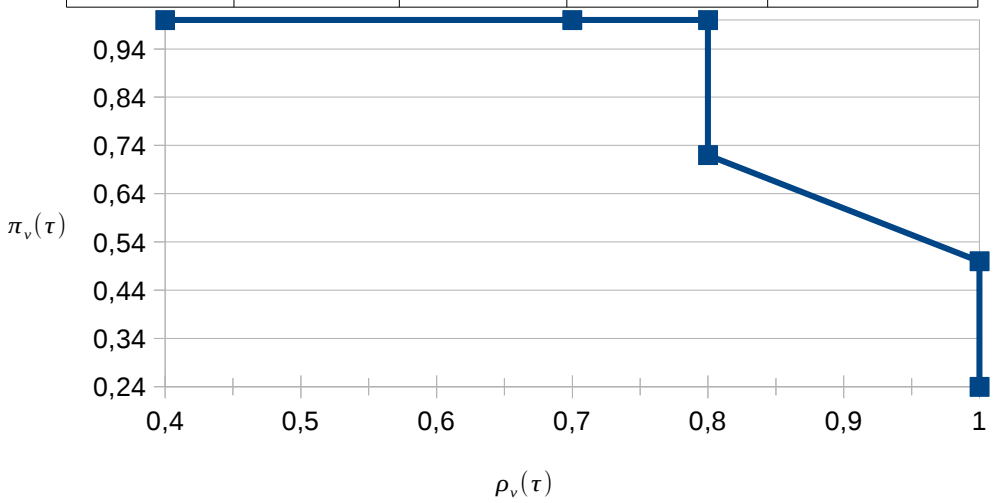
Consulta: “former || latter”

Elementos relevantes *(R_v)*: 10

Calculo de $\pi_v(\tau)$ y $\rho_v(\tau)$ para *τ = 0,0.2,0.4,0.6,0.8,1:*

τ	$\lvert D_v(\tau)\rvert$	$\lvert H_v(\tau)\rvert$	$\pi_v(\tau)$	$\rho_v(\tau)$
0	41	10	0,24	1
0,2	20	10	0,5	1
0,4	11	8	0,72	0,8
0,5	8	8	1	0,8
0,6	7	7	1	0,7
0,8	4	4	1	0,4
1	0	0	0	0

Curva de precisión/exhaustividad:



Precisión media interpolada: 0.922

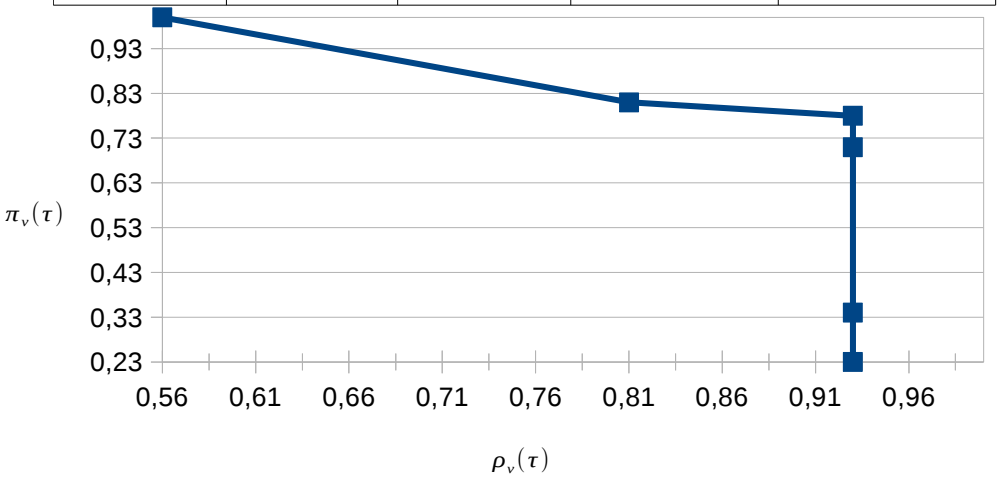
Consulta: “first || last”

Elementos relevantes *(R_v)*: 16

Calculo de $\pi_v(\tau)$ y $\rho_v(\tau)$ para *τ = 0,0.2,0.4,0.6,0.8,1:*

τ	$\lvert D_v(\tau)\rvert$	$\lvert H_v(\tau)\rvert$	$\pi_v(\tau)$	$\rho_v(\tau)$
0	63	15	0,23	0,93
0,2	44	15	0,34	0,93
0,4	21	15	0,71	0,93
0,5	19	15	0,78	0,93
0,6	16	13	0,81	0,81
0,8	9	9	1	0,56
1	0	0	0	0

Curva de precisión/exhaustividad:



Precisión media interpolada: 0.827

Precisión media interpolada entre las 7 consultas: 0.802.

Conclusión

Como vemos en las pruebas realizadas, las optimizaciones que hemos aplicado en este trabajo para mejorar la eficiencia y flexibilidad del sistema inicial muestran resultados positivos, permitiendo alcanzar la mejora objetivo en la viabilidad. Aun así, durante la implementación se han descubierto algunas mejoras que dadas las restricciones de tiempo no han llegado a ser implementadas y que queremos hacemos una mención a continuación para futuras actualizaciones.

Con la nueva implementación del proceso de indexación se ha logrado reducir el consumo de memoria de las colecciones a 4Gb, suficiente para el número actual. Sin embargo, como el tamaño del índice aumenta por cada libro, todavía existe un número máximo de colecciones que entran en memoria, para poder aumentar éste número sera necesario modificar el comportamiento del índice y utilizando un buffer, cargar a memoria solo las entradas del índice relevantes en el momento, aumentando probablemente el tiempo de respuesta del servidor pero eliminando la restricción al tamaño de la colección.

En el caso de las búsquedas conjuntas, aunque las aproximaciones utilizadas en la confianza de las palabras han obtenido resultados acorde a la intuición del usuario, se han utilizado funciones básicas de manipulación de cadenas para analizar la consulta, lo que dificulta trazar operaciones complejas y tratar con más operadores, como el OR o el NOT a nivel de linea. Una mejor práctica para una futura implementación debería definir un lenguaje formal y utilizar un analizador gramatical para resolver la consulta. Igualmente, las operaciones de conjuntos realizadas en el servidor web para resolver las búsquedas conjuntas, pueden ser reemplazas por funciones de maximización y minimización de probabilidades sobre las entradas del índice, que resultarían en un menor tiempo de consulta aunque también requiere implementar las mismas en el servidor del índice en lugar del servidor web.

Por último, aunque no esta incluido en el enfoque de éste trabajo, queremos remarcar que los buenos resultados conseguidos en la búsqueda de palabras clave basada en grafos de palabras viene de la rica información léxica y sintáctica que es explícitamente retenida en el grafo, sin embargo, palabras no indexadas, es decir no incluidas en el lexicón del

lenguaje, obtienen un resultado nulo, haciendo el enfoque inútil para este tipo de consultas. Debido a que en aplicaciones reales, es bastante probable con el tiempo muchas consultas caigan en ésta categoría, hemos dejado en la bibliografía ([11] y [12]) referencias a artículos de J. Puigcerver que presentan distintas soluciones a éste problema.

Bibliografía

[1] Transcritorium: http://transcriptorium.eu/

[2] Documentación C++: http://www.cplusplus.com/reference/

[3] Librería Boost-serialization: http://www.boost.org/doc/libs/1_59_0/libs/serialization/doc/

[4] Librería Libmicrohttpd: http://www.gnu.org/software/libmicrohttpd/

[5] A. H. Toselli y E. Vidal, transparencias texto manuscrito de la asignatura "Reconocimiento de Escritura" (RES) del master IARFID, Universidad Politécnica de Valencia, 2015.

[6] J. A. Sánchez, G. Mühlberger, B. Gatos, P. Schofield, K. Depuydt, R. M. Davis, E. Vidal, and J. de Does, "tranScriptorium: a European Project on Handwritten Text Recognition.," en el simposio ACM de ingeniería de documentos DOCENG, 2013, pp. 227-228.

[7] J. A. Sánchez, V. Bosch, V. Romero, K. Depuydt, and J. de Does, "Handwritten Text Recognition for Historical Documents in the tranScriptorium Project," en la primera conferencia internacional de acceso digital a herencias culturales textuales, New York, NY, USA, 2014, pp. 111-117.

[8] A. H. Toselli y E. Vidal, "Fast HMM-filler approach for key word spotting in handwritten documents," en la 12ª conferencia internacional en Análisis de Documentos y Reconocimiento, 2013, pp. 176-180.

[9] A. H. Toselli, E. Vidal, V. Romero, and V. Frinken, "Word-graph based keyword spotting and indexing of handwritten document images," Universidad Politécnica de Valencia, Tech. Rep., 2013.

[10] A. H. Toselli y E. Vidal, "Word-Graph based Handwriting Key-word Spotting: Impact of Word-Graph Size on Performance", en el 11º taller internacional en Sistemas de Análisis de Documentos (DAS), 2014, pp. 176-180.

[11] J. Puigcerver, A. H. Toselli, and E. Vidal, "Word-Graph-Based Handwriting Keyword Spotting of Out-Of-Vocabulary Queries," en la 22ª Conferencia Internacional en Reconocimiento de Patrones (ICPR), 2014.

[12] J. Puigcerver, A. H. Toselli, and E. Vidal, "Word-Graph and Character-Lattice Combination for KWS in Handwritten Documents," en la 14ª conferencia internacional en fronteras del reconocimiento de texto manuscrito (ICFHR), 201