Explorador de Tópicos Científicos

Reporte Técnico de la Versión 1.0

Índice de Contenidos

1. Introducción

Una *tag cloud*, también conocida como nube de palabras, es una valiosa forma de representar información a través de un posicionamiento y uso de colores que resulten atractivos para quien lo aprecia. A pesar de que se acostumbra a darle un uso meramente artístico, tal y como lo hace la herramienta Wordle™, se observó una área de oportunidad en el uso de una nube de palabras para transmitir información pertinente sobre algún tema.

El uso de nubes de palabras para dar una primera vista a cierto tema resulta útil, pues a partir de la impresión general del usuario, él decide si desea conocer más acerca del tema plasmado en la nube de palabras. Descrito de otra manera, una nube de palabras es el punto de entrada para profundizar acerca de un tema de interés.

A pesar de que existen herramientas para crear nubes de palabras dado un escrito, están muy limitadas las opciones que integran un sistema de clasificación de información por temas a través de una nube de palabras.

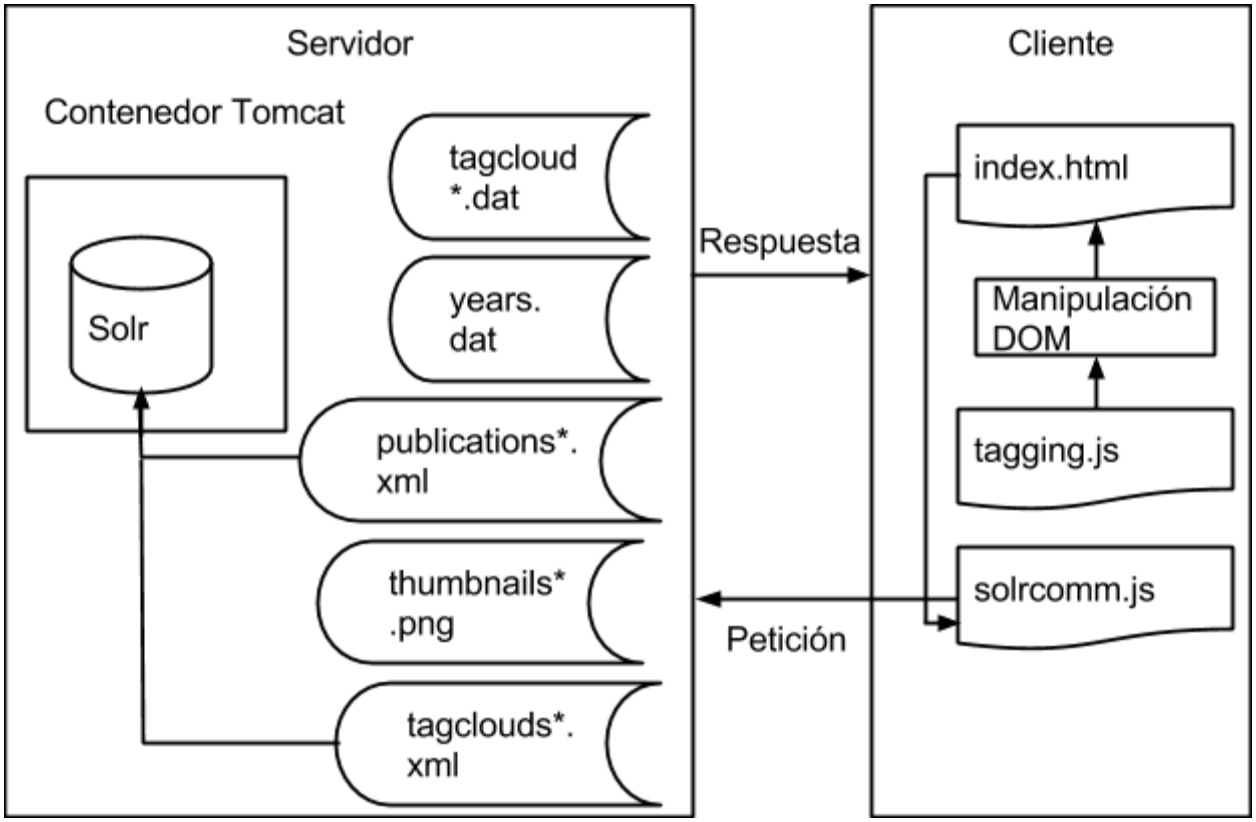
2. Objetivos

El objetivo general es el diseñar y construir un sistema con una arquitectura que integre la información de diversos temas en una colección de nubes de palabras, de tal forma que se logre hacer una búsqueda sobre algún tema de manera inteligente, considerando la importancia individual de cada término.

De manera particular, la intención es que la visualización de las nubes de palabras resulte atractiva para el usuario, e igualmente sea posible distinguir características relevantes de los términos presentados, así como el obtener más información al respecto a través de interacción con la nube de palabras.

Entre los requisitos no funcionales, se busca el que haya un alto grado de portabilidad de acceso al sistema; es decir, que funcione en equipos con distintos sistemas operativos y paquetería instalada.

3. Arquitectura del Sistema (vista general)



El esquema anterior describe qué componentes existen en el sistema y cómo es la interacción entre ellos. En el lado del servidor, se mantienen los archivos requeridos para organizar el catálogo de nubes de palabras. Los documentos xml están categorizados en dos: en un grupo están aquellos que contienen información sobre cada nube de palabras, incluyendo términos, nombre del *thumbnail* o vista preliminar, nombre del archivo con la estructura de la nube, y el nombre del archivo con los años de actividad de cada término. En otro grupo de documentos xml está la información de las publicaciones o artículos incluidas en el catálogo. Ambas clasificaciones son añadidas al índice de Solr, aplicación que está contenida dentro de Tomcat.

En el lado del cliente, está el documento HTML que sirve como interfaz gráfica de usuario, acompañada de programas script de Javascript. *tagging.js* se encarga de crear y modificar los elementos necesarios para mostrar una nube de palabras, mientras que *solrcomm.js* tiene las funciones para establecer un enlace de comunicación con la aplicación de solr, y así hacer consultas respecto algún término del que se busca obtener una nube de palabras, o bien, para conocer más información acerca de una etiqueta dentro de una nube de palabras. El manejo dinámico de la información es logrado a través de la manipulación del DOM del documento *index.html*.

4. Herramientas

Herramienta, Tecnología o API	Aplicación
HTML5 y Javascript	Proporciona las utilidades para crear el front end y permite cierto grado de procesamiento de información, así como el manejo del DOM *en el vuelo*.
Solr	Aplicación o plataforma web que gestiona índices de documentos para una búsqueda eficiente y rápida. Requiere de un esquema para los documentos a ingresar al índice.
HTML5 Canvas	Permite la creación y manipulación de píxeles en una página web.

5. Creación de una Nube de Palabras (*front end*)

5.1 Lectura de datos

Se comenzó con la suposición de que los datos requeridos para formar una nube de palabras sobre un tema ya estaban disponibles y preprocesados. Es decir, se contaba con un archivo CVS (comma separated values) que detalla la información: palabra, tamaño, coordenadas x_1, y_1, x_2, y_2.

5.2 Definición del Canvas

En lo que concierne a la creación individual de una nube de palabras, se utilizó el lenguaje HTML5 y una de sus API: Canvas. A través de esta API es posible delimitar un espacio para hacer dibujos y manipulación de píxeles. En este caso, se utilizó el API para *pintar* las palabras en los pixeles del espacio proporcionado.

El canvas dentro de una página válida HTML se define como:

```
<canvas id=”lienzo”></canvas>
```

Fue relevante especificar el id del elemento canvas, pues permite su manipulación a través del DOM usando Javascript.

```
    canvas = document.getElementById("lienzo"),
    context = canvas.getContext('2d');
```

```
    canvas.width = 1100;
    canvas.height = 400;
    context.textBaseline = "top";
```

Código 1.1 describiendo la inicialización del canvas y el contexto

Para inicializar el canvas, es necesario recuperar tal elemento del DOM utilizando la función `document.getElementById()` y después se extrae el *contexto* del canvas, es decir, el sistema de coordenadas definido para ese espacio. El reajuste del *text baseline* permite dibujar elementos empezando de arriba a abajo, y de izquierda a derecha, lo cual es lo más apropiado para este caso donde ya se cuentan con las coordenadas de cada palabra.

5.3 Escribir texto en Canvas

Una función sencilla para *pintar* texto en un canvas es la siguiente:

```
function stampText(tag, x, y, size, color, font) {
    context.font = size + 'px ' + font;
    context.fillStyle = color;
    context.fillText(tag, x, y);
}
```

Código 1.2 Escribiendo texto en el canvas

El código 1.2 fue el que en un inicio se utilizó para crear la nube de palabras, pues se requería de la palabra, su tamaño, posición en x,y, color, y tipo de letra.

5.4 Definición de color gradiente

Después de una corta iteración de desarrollo y diseño, surgió el requerimiento de crear colores gradientes sobre los términos en la nube de palabras. La intención con los colores gradientes era el poder distinguir qué términos dentro de una nube de palabras han tenido actividad reciente, cuáles tuvieron su auge en el pasado, y cuáles se han mantenido constante en actividad durante los años. Para construir una solución que incluya una definición de un gradiente, se exploró más profundamente la API de canvas que permitiera hacer esto.

El gradiente se puede aplicar a través de la función `context.createLinearGradient(x1, y1, x2, y2);` la cual requiere las cuatro esquinas donde se aplicará el gradiente. En el caso de la nube de la palabras, las esquinas constituirían aquellas dadas por cada palabra individualmente.

Se deben definir puntos de cambios del gradiente, es decir, puntos entre 0 y 1 que describa los colores para una suave transición.

```
    var grad = context.createLinearGradient(x1,y1,x2,y2);
    grad.addColorStop(0, '#213325');
```

```
    grad.addColorStop(0.7, '#80C892');
    grad.addColorStop(1, '#6CE9AC');
    context.fillStyle = grad;
    context.fillText(word, x, y);
```

Código 1.4 Añadiendo cambios de color en el gradiente y dibujando el texto

5.5 Estructura de datos para etiquetas

Considerando que hay una considerable cantidad de palabras a incluir a una única nube de palabras, se utilizó un enfoque orientado a objetos para crear una estructura que contuviera los datos relevantes para cada etiqueta de la nube. Este objeto es definido de la siguiente forma:

```
function Tag(word, size, x, y, years) {
    this.word = word;
    this.size = size;
    this.x = x;
    this.y = y;
    var endX, endY, link;
    this.years = years.sort(function(a, b) {
    return a-b
    });
}
```

Código 1.5 Definiendo la estructura del objeto Tag

5.6 Modificación de gradiente relativo a años de actividad

Para propósitos de definición de las transiciones de colores en cada gradiente, es requerido un arreglo de años, el cual hace referencia a la época en donde ha estado en actividad tal palabra.

En el código 1.4 hay tres cambios de colores; sin embargo, la intención es que el número de transiciones entre gradientes sea definido por los años de actividad de cada término. El comportamiento esperado es que entre más activo en la literatura haya sido un término dado, se apreciarán tonalidades más claras en el gradiente. De otra manera, si los años son de más antigüedad, el gradiente tendrá tonalidades más opacas. Dado un vector ordenado con los años donde aparece un término, el cálculo del valor RGB se hace de la siguiente manera:

```
  for(var i=0; i<tag.years.length; i++) {
    intensity = Math.abs(Math.pow(((baseYear - tag.years[i])/100), 0.7));
    colorValue = parseInt(Math.floor((1.0-intensity)*255));
    color = 'rgb(' + 120 + ', ' + colorValue + ', ' + colorValue + ')';
  }
```

Código 1.6 Algoritmo para generar puntos de cambio de color

En el código 1.6, es relevante mencionar la fórmula $\left|\left(\frac{año\ base - año}{100}\right)^{0.7}\right|$ (Schaeffer, 2013), pues toma en cuenta un año base, y genera un valor relativo entre el año base y el año en cuestión. Este valor se usa para crear el código de color RGB (120 es un valor arbitrario para darle cierto color general al gradiente). El valor del RGB es usado para cada punto de cambio de color, y el número de cambios es definido por el número de años disponibles para un término en particular.

Fue importante considerar que hay etiquetas en algunas nubes de palabras donde sólo hay 1 o 2 años asociados a la palabra, por lo que no se puede crear un gradiente apropiado. Para estos casos, el color será uniforme, aunque sí se verá afectada la opacidad o luminosidad de la palabra dependiendo del año reportado.

6. Índice para organización de Nubes de Palabras (*backend*)

6.1 Elección entre índice y base de datos relacional

Un índice es una estructura que permite almacenar y organizar documentos en diversos campos o características que contengan estos. En una base de datos relacional, hay claves únicas relacionadas a columnas que indican alguna característica.

En el caso de este sistema, lo conveniente es poder hacer búsquedas completas de texto sobre uno o más campos. Comparando motores de búsqueda de texto completo que trabajan sobre índices y bases de datos relacionales que implementan una funcionalidad para búsqueda sobre texto, se ha concluido que es más conveniente utilizar la estructura del índice y un motor de búsqueda. Entre muchas de las opciones consideradas (Lucene, ElasticSearch), se ha decidido utilizar Solr, la cual cuenta con amplia documentación y soporte para aplicaciones web.

6.2 Introducción a Solr

Solr es una plataforma de búsqueda de código abierto, cuyas funcionalidades incluyen búsqueda de completa de texto, resaltado de coincidencias, *faceted search*, indexamiento de casi tiempo real, entre otras. Está basado en la librería Lucene para búsquedas sobre índices, con la diferencia que Solr es una *plataforma* lista para utilizarse.

La instalación y configuración de Solr para el contenedor Tomcat se hizo siguiendo tutoriales de Greg Somers (2013) y José González (2013).

6.3 Definición del *schema*

Para poder agregar documentos al índice, fue necesario definir un *schema* donde se especifica qué clase de información irá a cada campo del documento, y si esta información puede ser

buscada y almacenada para recuperarse. Existen muchos tipos de campos que se pueden elegir para el texto, pero los definidos fueron los siguientes:

```
<field name="texto_completo" type="text_general" indexed="true" stored="true"
multiValued="false" termVectors="true" termPositions="true"
termOffsets="true"/>
<field name="cloud" type="string" indexed="false" stored="true"/>
<field name="thumbnail" type="string" indexed="false" stored="true"/>
<field name="cloudyears" type="string" indexed="false" stored="true"/>
<field name="authors" type="string" indexed="true" stored="true"
multiValued="true"/>
<field name="year" type="tint" indexed="false" stored="true"/>
<field name="journal" type="string" indexed="false" stored="true"/>
```

6.4 Agregar documentos al índice

Los documentos se pueden agregar manualmente desde la pantalla de administración de Solr, pero esto no resulta eficiente para grandes cantidades de documentos. Por tal motivo, se utilizó el programa escrito en Java *post.jar* para agregar documentos al índice. También está la una versión escrita en bash de nombre *post.sh*. Para el caso de la versión de Java, se ejecuta de la siguiente manera:

```
java -Durl=http://MYSERVER-URL:PORT/solr/update -jar post.jar document.xml
```

Donde MYSERVER-URL es la URL del servidor donde está Solr, y 8080 es el PORT es el puerto apropiado (eg. 8080).

7. Consultas al índice desde Página Web

7.1 Función para consulta a Solr

XMLHttpRequest es una API disponible en Javascript que es utilizada para enviar peticiones HTTP o HTTPS a un servidor web, y almacenar la respuesta como datos en el programa. Dada esta API, no fue necesario emplear otro lenguaje de programación más robusto que implemente un medio de comunicación entre el programa y Solr.
La función diseñada para hacer peticiones al servidor web que tiene Solr fue la siguiente:

```
function queryRequest(url, params) {
    var myRequest = new XMLHttpRequest();
    myRequest.open('POST', url, true);
    myRequest.setRequestHeader('Content-Type',
'application/x-www-form-urlencoded');
    myRequest.onreadystatechange = function() {
      if (myRequest.readyState == 4) {
      updatePage(myRequest.responseText);
```

```
      }
    }
    var strData = params.join('&');
    myRequest.send(strData);
}
```

En el código anterior se empleó la programación asíncrona, pues sólo se toma acción cuando la respuesta llega y está lista para ser procesada. En este caso, la función que se usó fue `updatePage(myRequest.responseText)` la cual se encarga de modificar los elementos de la página con la respuesta recién recibida.

7.2 Estructura JSON de la respuesta

Dependiendo de los parámetros añadidos a la petición del cliente hacia el servidor, es posible que el formato de respuesta se dé en forma de una estructura JSON. Esta estructura puede ser accedida nativamente por Javascript, y así obtener las partes que se buscan con facilidad.
Por ejemplo, para obtener el número de coincidencias o *hits* en la consulta al índice, se usó:

```
var jsonQuery = JSON.parse(msg);
var numFound = jsonQuery.response.numFound;
```

7.3 Ranking de resultados con TF-IDF

Considerando que existen términos que son más importantes que otros dependiendo del número de veces que aparece en ese documento (nube de palabras) y en otros, se dejó abierta la opción de poder hacer la consulta al índice usando tf-idf, es decir, *term frequency-inverse document frequency.*

La notación de la consulta se debe cambiar la siguiente manera para que sea posible hacer el ranking considerando TF-IDF.

```
&q=mul(0.1,mul(tf(texto_completo,word),idf(texto_completo,word)))
```

Donde *word* es el término que se ingresa para buscar.

8. Estructura de la Aplicación Web (*front end)*

8.1 Elementos del documento HTML

La página HTML está diseñada de tal manera que toda actividad que ocurra en el sistema se vea reflejada sobre la misma página. Por tal motivo, se definen todos los elementos a requerir en cualquier momento, aunque en un principio están vacíos. El contenido dentro de la etiqueta *body* es el siguiente:

```
 <h1 class="headline">Explorador de T&oacute;picos de
Investigaci&oacute;n</h1>
  <div class="formato">
    <form id="forma" name="qform" onsubmit="queryRequest(location.protocol +
'//' +location.host + '/solr/collection1/select/', null); return false;">
      <fieldset>
        <legend>Buscar por</legend>
      </fieldset>
      <span><input name="query" type="text" class="search square" ><input
type="submit" value="Buscar"></span>
    </form>
  </div>
    <div id="b">
      <div id="infobox"></div>
      <div id="result"></div>
      <div id="tagcloud"></div>
    </div>
```

Parte relevante de la estructura anterior es que se incluye un espacio de *input* de texto para capturar el término del cual se quieren buscar nubes de palabras. También fue importante hacer una división con la etiqueta *div* para manejar mejor el contenido: dentro de una etiqueta padre están las secciones de *infobox* (información de la publicación), *result* (lista de resultados encontrados en forma de thumbnail) y *tagcloud* (sección reservada para el canvas).

8.2 Manejo del DOM

Para actualizar la página web con la respuesta recibida del servidor web, fue necesario modificar el DOM utilizando Javascript. Por ejemplo, para el caso donde se necesita poner un título por encima de la nube de palabras, se haría lo siguiente:

```
function updateTitle(allTags) {
    ...
    var titleTag = '';
    var cloudDiv = document.getElementById('tagcloud');
    var cloudCanvas = document.getElementById('lienzo');
    ...
    var textNode = document.createTextNode(titleTag);
    var titleNode = document.createElement("H2");
    titleNode.appendChild(textNode);
    cloudDiv.insertBefore(titleNode, cloudCanvas);
}
```

En el código anterior, el nodo de texto creado se añade a la etiqueta *H2*, la cual es insertada dentro de la etiqueta con id *tagcloud*, pero antes de la etiqueta *lienzo*, la cual representa el canvas donde se dibuja la nube de palabras.

8.3 Presentación de resultados en *thumbnails*

Para facilitar la vista de los resultados a una consulta, se decidió que se podrían previsualizar en forma de thumbnails, es decir, imágenes pequeñas de las nubes de palabras encontradas. Aunque en un principio se pensó apropiado usar alguna herramienta que de manera automatizada creará imágenes de los documentos HTML que muestran una nube, se decidió en utilizar los documentos post script (.eps) con los que ya se contaban. El único paso adicional fue el convertirlos a imagen png con el programa *ImageMagick* y usando el comando:

```
convert tagcloud.eps tagcloud.png
```

Dentro de la respuesta en formato JSON de Solr, vienen varios campos que se pueden usar para listar los resultados en forma de catálogo de *thumbnails*.

```
var jsonQuery = JSON.parse(msg);
var item;
var newThumbnail = null;
var theDiv = document.getElementById("result");

for(var i=0; i<jsonQuery.response.docs.length; i++) {
item = jsonQuery.response.docs[i];

newThumbnail = document.createElement("img");
newThumbnail.setAttribute('src', item.thumbnail);
newThumbnail.setAttribute('width', '33%25');
newThumbnail.setAttribute('height', 'auto');

theDiv.appendChild(newThumbnail);
}
```

En el código anterior, se recorren todos los documentos de la respuesta en JSON, y para cada una, se toma el valor de *thumbnail,* el cual es el nombre de la imagen correspondiente a ese resultado. Después de crear una etiqueta *img* para cada documento y agregarle la dirección de la imagen, se agrega al *div* donde se concentran los resultados.

8.4 Interacción con la Nube de Palabras

Una vez desplegada la nube de palabras con los colores de gradiente apropiados, es posible *interactuar* con ésta a través de clics del ratón. La premisa es poder conocer acerca de alguna publicación donde aparece en el título la etiqueta a la que se le dio clic. Para poder reconocer el clic dentro del canvas se asigna el evento *mousedown a éste*, dentro del cual se obtiene la posición x,y:

```
canvas.addEventListener("mousedown", function(event) {
  var mousePos = getPosition(canvas, event);
```

```
}
```

Una vez obtenidas las coordenadas, se recorre el arreglo de etiquetas que conforman la nube de palabras, comparando ambos conjuntos de posiciones, hasta encontrar el que corresponde a la etiqueta seleccionada.

Habiendo reconocido de qué etiqueta se trata, se usa la interfaz XMLHttpRequest para hacer otra petición a Solr dentro del servidor web, sólo que en este caso no se buscaría sobre *texto_completo*, es decir, el conjunto de palabras que forman a la nube de palabras, sino el campo *name*, pues es donde se almacena el nombre para cada publicación incluida en la colección. Obtenida una respuesta con el conjunto de publicaciones que corresponden a la etiqueta buscada, se puede crear una tabla de información adicional.

8.5 Tabla con Información de la Publicación

Para las situaciones en las que el usuario requiera más información sobre algún ítem de la nube de palabras, es conveniente mostrar información sobre una publicación en cuyo título aparezca tal palabra.

A través de la manipulación del DOM, es posible desplegar una tabla hecha con etiquetas HTML *en el vuelo*, conteniendo información pertinente sobre la publicación encontrada. Utilizando un ciclo *for* para la estructura JSON resultante, se hace una tabla de dos columnas y los renglones: título, autores, año, y revista.

Autores	oscar h. ibarra,sara woodworth
Título	characterizing regular languages by spiking neural p systems.
Año	2007
Revista	int. j. found. comput. sci.

9. Capturas de Pantalla

A continuación se muestran algunas capturas de pantalla del sistema, en el lado del cliente, para mostrar la funcionalidad desarrollada.

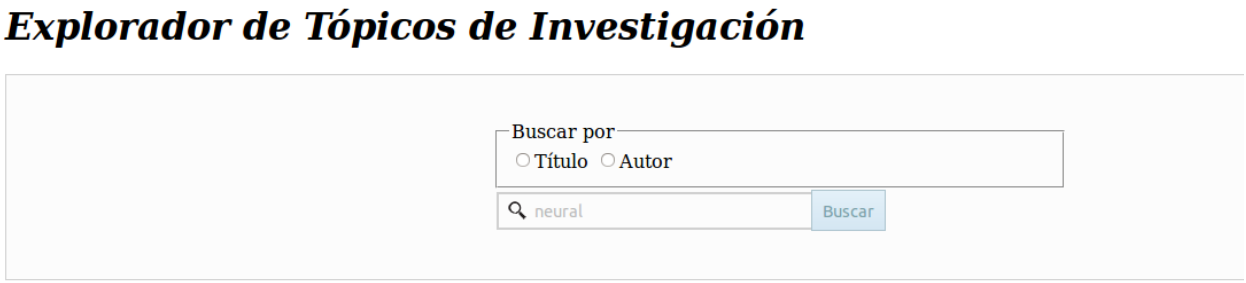


Figura 1. Página Inicial

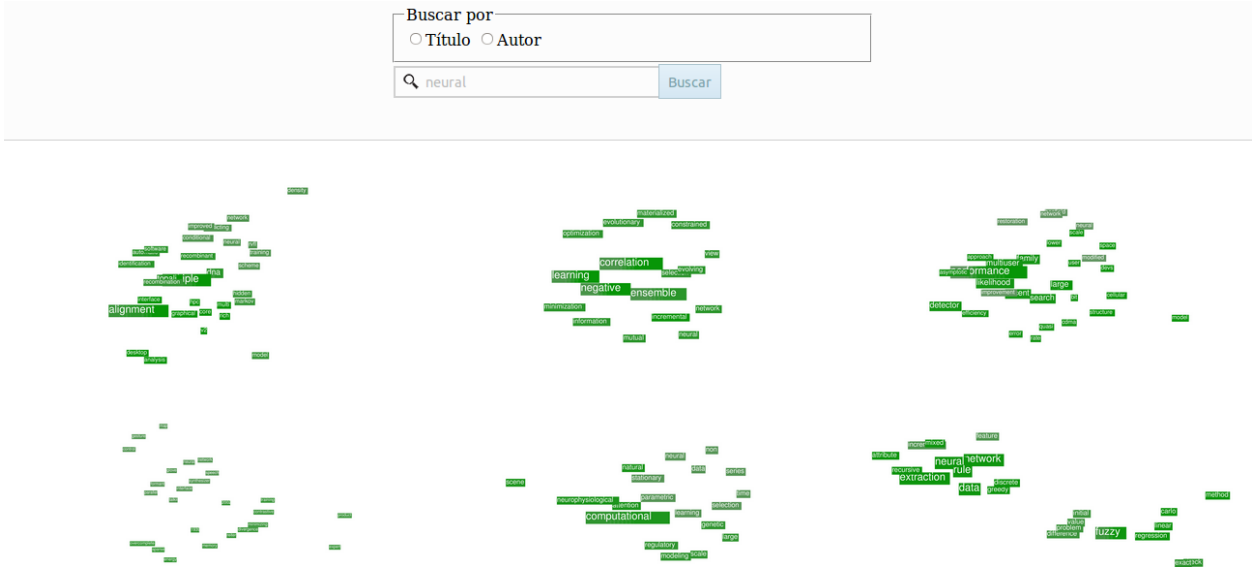


Figura 2. Resultados mostrados en *thumbnails*

negative. learning. correlation. ensemble.

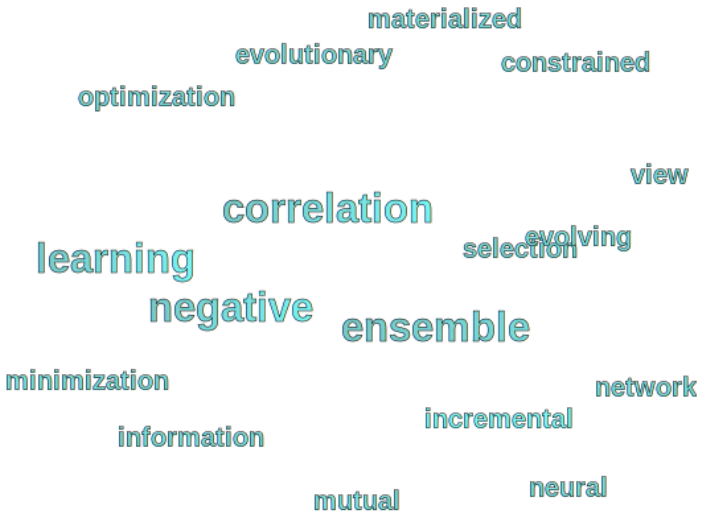


Figura 3. Una de las nubes de palabras asociadas a "*neural*" incluyendo título y gradientes

Información de la Publicación

Autores	oscar h. ibarra,tao jiang
Título	learning regular languages from counterexamples.
Año	1991
Revista	j. comput. syst. sci.

Figura 4. Información adicional de una publicación con la palabra *learning* en su nombre

evolutionary. lda. weighted. design.

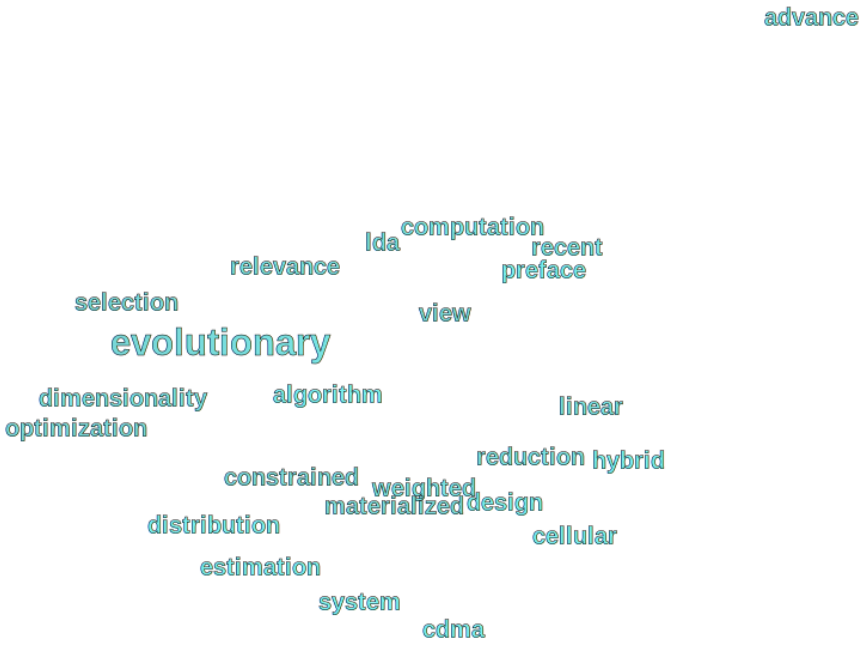


Figura 5. Nube de palabras al buscar el término *data*

Autores	sheng zhong
Título	efficient, anonymous, and authenticated conference key setup in cellular wireless networks.
Año	2008
Revista	computers a electrical engineering

Figura 6. Publicación encontrada al buscar *cellular*

10. Conclusión

Se propuso e implementó una arquitectura para un sistema web y portable apoyado de la plataforma Solr, propicio para la búsqueda de temas visualizadas en nubes de palabras. Las nubes de palabras fueron diseñadas de tal manera que en primera vista se diera información relevante a conocer, tal como la importancia de ciertas palabras, así como la relación entre ellas. Además, se implementó un esquema de gradientes donde el color más claro significaba más actividad reciente, y el más opaco, actividad de más antigüedad. Por último, se implementó

el medio en el que el usuario puede interactuar con la nube de palabras para conocer más acerca de la publicación donde aparece tal palabra de interés.

11. Referencias

Apache Software Foundation. *Apache Solr*. http://lucene.apache.org/solr/

González, José. Agosto 2013. *Instalar y configurar Apache Solr 4.4.0 con Tomcat7 en Ubuntu Server 13.04.* http://proyectosbeta.net/2013/08/instalar-y-configurar-apache-solr-4-4-0-con-tomcat-7-en-ubuntu-server-13-04/

Schaeffer, Elisa (comunicación personal, 27 de oct., 2013) explicó una fórmula para obtener valores resultantes de años relativos a un año base para obtener un gradiente de actividad reciente o antigua.

Somers, Greg. Julio 2013. *Install apache Solr 4.4 on Ubuntu 12.04 with Tomcat 7 and MySql Data Import.* http://gregsomers.com/blog/2013/07/30/install-apache-solr-44-on-ubuntu-1204-with-tomcat-7-and-mysql-data-import