elasticsearch

Jesús García Minas
Universidad de Oviedo
Curso 2017/18

Índice:

Introducción

Elastic Search es un servidor de búsqueda desarrollado en java que permite realizar consultas a una colección de documentos indexados utilizando un estándar como JSON lo que permite que se trate de un programa muy simple con el que aprender y trabajar.

Por comodidad, se incorpora cerebro que permite realizar una serie de búsquedas de manera más cómoda ya que realizándolas de manera estándar se complicaría su uso lo cual no es recomendable en este ámbito de tipo pedagógico.

Programa

La información utilizada para la realización del trabajo será la misma que se ha utilizada en clase, los tweets del 2 al 4 de febrero que coincidieron con la Super Bowl lo que permite una variedad de búsquedas que permiten una amplia cantidad de búsquedas.

Se ha filtrado con grep los tweets para que solo aquellos que estén escritos en inglés aparezcan ya que facilita las búsquedas tener solo un idioma.

El programa tratará de realizar expansión de consultas de manera manual, es decir, cogerá aquellos tweets válidos e irá almacenando las palabras que salen junto a la palabra buscada.

Existen un número de palabras que se filtrarán en inglés ya que no aportan ningún valor a la búsqueda. Este tipo de palabras se las conoce como “palabras vacías”. Se utilizará el archivo .txt visto en clase, pero se ha modificado eliminando todos los comentarios y los espacios en blanco para que la lectura en php se realice de una forma más cómoda.

Las palabras resultantes serán evaluadas mediante un algoritmo denominado N.G.D o “Normalized Google Distance” cuyo resultado muestra como de “inseparables” son esos 2 términos dando un resultado más próximo a cero cuanto más cerca estén uno del otro.

Esta funcionalidad de añadir sinónimos a la consulta la incorpora elastic search con simple expansión, simple contraction y genre expansión. En la simple expansión, en caso de hacerlo durante la indexación tiene también desventajas ya que al tener una lista con todos los sinónimos de las palabras lo que provoca que tendrán que estar todos indexados. Otro inconveniente es que todas las palabras tienen el mismo IDF (Es decir, que se valoran de igual manera todos los términos lo que no permite “decidir” al programa que término puede ser más util). Simple contraction consiste en lo contrario, a partir de un conjunto de palabras buscará un sinónimo que permita obtener una búsqueda similar. Finalmente, Genre Expansion busca sinónimos que son genéricos de una

palabra. Por ejemplo, un sinónimo genérico de fútbol sería deporte, de gato sería mascota, etc.

A continuación mostraré capturas del programa realizado donde iré mostrando paso por paso como he realizado el problema planteado.

```php
$client = Elasticsearch\ClientBuilder::create()->build();

$cons = "football"; //Término a buscar

$vacias=file("stop.txt", FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES); //Array palabras vacias

$diccionario = array(); //Diccionario palabras

$ngdArray = array(); //Diccionario palabras candidato ngd

$vacio = false; //Es una palabra vacía?

$numeroResultados = 1000; //Número de resultados a evaluar sus palabras

$palabras = 10; //Número de palabras con mas aparición a evaluar

$cuentaX = 0; //Resultados consulta original

$cuentaY = 0; //Resultados palabras a evaluar

$cuentaXY = 0; //Resultado conjunto palabra + consulta

$numeroTotal = 394990; //Número de tweets

$palabrasFinales = 5; //Número de palabras máximo a introducir en la consulta

$text = ""; //Texto de cada tweet

// 2008-feb-02-04-en/_search?q=text:$cons
//
$params = [
    "index" => "2008-feb-02-04-en",
    "type" => "tweet",
    "body" => [
    "size" => "1000",
        "query" => [
            "match" => [
                "text" => $cons,
            ]
        ]
    ]
];

$results = $client->search($params);
```

En primer lugar se inicializan los atributos con los que trataré de obtener los sinónimos de la palabra a buscar. Los comentarios explican en detalle la función de cada uno de estos.

```
foreach ($results as $res) {
    $res = $res['hits'];
    for ($i = 0; $i < sizeof($res); $i++) {
        $resultados = $res[$i];
        $resultados = $resultados['_source'];
        if ($numeroResultados>0) {
            $text = $resultados['text'];
            $textos = split(" ", $text);
            foreach ($textos as $t) {
                foreach ($vacias as $v) {
                    if ($v==strtolower($t) || $t=="" || $t=="-" || $t=="." || $t=="(!)") {
                        $vacio = true;
                    }
                }
                if (!$vacio && strtolower($t)!=strtolower($cons)) {
                    $diccionario[(string)$t] = ($diccionario[(string)$t] + 1);
                }
                else {
                    $vacio = false;
                }
            }
            $numeroResultados--;
        }
        else {
            break;
        }
    }
}
```

Este conjunto de bucles anidados se encargaran de coger los primero n términos encontrados y contará todas aquellas palabras que no sean términos vacíos que aparecen en el documento .txt utilizado.

```
$cuentaX = sizeof($res);
foreach ($diccionario as $key => $value) {
    if ($palabras>0) {
        $params = [
            "index" => "2008-feb-02-04-en",
            "type" => "tweet",
            "body" => [
            "size" => "1000",
                "query" => [
                    "match" => [
                        "text" => $key,
                        ]
                    ]
                ]
            ];

        $results = $client->search($params);
        foreach ($results as $res) {
            $res = $res['hits'];
            $cuentaY = sizeof($res);
        }

        $consulta = $cons;
        $consulta .= " ";
        $consulta .= $key;

        $params = [
        "index" => "2008-feb-02-04-en",
        "type" => "tweet",
        "body" => [
            "query" => [
                "match" => [
                    "text" => [
                        "query" => $consulta,
                        "operator" => "and"
                        ]
                    ]
                ]
            ]
        ];

        $results = $client->search($params);
        foreach ($results as $res) {
            $res = $res['hits'];
            $cuentaXY = sizeof($res);
        }

        $numerador = max(log($cuentaX), log($cuentaY))-log($cuentaXY);
        $denominador = log($numeroTotal) - min(log($cuentaX), log($cuentaY));
        $ngd = $numerador / $denominador;
        $ngdArray[$key] = $ngd;
        $palabras--;
```

En esta segunda parte se harán consultas con los términos con más resultados y se guardará el número de resultados encontrados (Máximo 1000). Posteriormente se utilizará la fórmula de la N.G.D:

$$\mathrm{NGD}(x,y) = \frac{\max\{\log f(x), \log f(y)\} - \log f(x,y)}{\log N - \min\{\log f(x), \log f(y)\}}$$

N : Total number of web pages searched multiplied by the average number of words per page

f(x) : Number of web pages with word *x*

F(y) : Number of web pages with word *y*

f(x, y) : Number of web pages with both *x* and *y*

Y se almacenará el resultado en otro diccionario.

```php
asort($ngdArray);
print_r($ngdArray);

foreach ($ngdArray as $key => $value) {
    if ($palabrasFinales>0) {
        $cons .= " ";
        $cons .= $key;
        $palabrasFinales--;
    }
}

$params = [
    "index" => "2008-feb-02-04-en",
    "type" => "tweet",
    "body" => [
    "size" => "1000",
        "query" => [
            "match" => [
                "text" => [
                    "query" => $cons,
                    "operator" => "or"
                    ]
                ]
            ]
        ];
    $results = $client->search($params);
    //print_r($results);
?>
```

Finalmente se procedería a añadir a la consulta los mejores términos encontrados resultando una consulta más completa.

Consultas (10 términos similares)

- **New England Patriots:** New, Patriots, Giants, York, Date, (Reuters), super, bowl, vs, go
- **New York:** New, Ago by, times, york, giants, los, 5, england, google
- **New York Giants:** York, giants, new, Glendale, (Reuters), england, 17-14, times, hours, ago by
- **NFL:** new, achieved, york, giants, victory, history, 17-14, one, england, super
- **Plaxico Burress:** burress, plaxico, plaxico?, giants, giants', super, practice, ->, can, manning
- **Quarterback:** vick, jailed, eli, manning, Brady, tom, can, keep, 1, hour
- **Randy Moss:** Randy, Moss, Moss., td, Brady, will, just, one, game, get
- **running back:** back!, back., back, running, errands, just, work, home, new, welcome.
- **Super Bowl:** bowl., bowl!, bowl, super, watching, giants, party, xlii, commercials, go
- **University of Phoenix:** university, Phoenix, washers, available, york, dryers, (Reuters), new, two, available.
- **Wide receiver:** &, wide, awake, post, really, open, world, awake, new, blog
- **XLII:** xlii, super, bowl, giants, new, patriots, superbowl, york, england, win

Tras la realización de estas búsquedas he podido llegar a varias conclusiones acerca de cómo se podría mejorar dicho programa con un conocimientos de gramáticas superior. Por una parte, los números no deberían ir solos, ya que necesitas una unidad para saber de qué estás hablando. Por ejemplo, en New York sale un 5, pero no sabes si se refiere a un número de un jugador, a los minutos que faltan o a otra cosa. Por otra parte, tienes el 17-14 que si es significativo al verse que es el resultado final. Deberían eliminarse todos los signos de puntuación que repiten palabras ya que el programa está hecho simplemente quitando los espacios de las palabras por lo que aparecen resultados repetidos en las búsquedas. En las palabras compuestas no aporta nada de valor encontrar resultados con sus palabras simples, ya que en el caso de New York Giants, quizás de Giants si es útil New York pero evidentemente New y York por separado no aportan nada de valor. Por último, en el caso de los números que he escogido, 10 sinónimos son demasiados, ya que no hay suficiente muestra para llegar a tantos sinónimos según los números que he visto, entre 5 y 7 sería lo óptimo para este ejemplo, sin embargo, esto no quita que para una aplicación con más índices, más palabras en el diccionario… etc, si podrían utilizarse más sinónimos quitando además todos los errores comentados anteriormente.

Conclusiones

A pesar de que en un principio Elastic Search puede parecer desde fuera una herramienta complicada de utilizar, tras utilizarla durante este trabajo mi punto de vista sobre esta ha cambiado a mejor.

Por una parte, su posibilidad de ser utilizada por prácticamente cualquier lenguaje la hace una herramienta de fácil acceso además de no necesitar grandes conocimientos de lenguaje para empezar a mejorar las consultas (Aunque evidentemente habrá un aumento de dificultad de forma gradual en caso de necesitar realizar consultas más complejas que aumenten la complejidad del programa como puede suceder con la N.G.D en caso de introducir números de palabras muy elevados como ha sido mi caso al principio ya que produce un elevado tiempo de espera que en un buscador comercial no se podría tolerar).

Por otra parte, la utilización de ficheros .JSON como fuente de información, lo convierte en un programa que acepta un formato que se utiliza para prácticamente cualquier cosa. En mi caso, el formato .JSON fue muy útil para encontrar una base de datos para xQuery ya que la conversión de .JSON a .XML es también muy sencilla.

Finalmente, si tenemos un programa que recoge información de un estándar utilizado muchísimo a nivel mundial y que acepta cualquier lenguaje de programación para poder hacer los cambios que tu desees, tenemos una herramienta muy potente en la que en poco tiempo puedes analizar un fichero .JSON enorme (Como en el caso de las prácticas con casi 400.000 líneas de código).

Bibliografía

https://es.wikipedia.org/wiki/Elasticsearch

http://php.net/manual/es/

https://www.elastic.co/guide/en/elasticsearch/guide/current/synonyms-expand-or-contract.html

https://en.wikipedia.org/wiki/Normalized_Google_distance