ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

“Implementación de visualizador de evolución de tópicos extraídos de plataforma de microblogging.”

INFORME DE PROYECTO INTEGRADOR

Previo a la obtención del Título de:

INGENIERO EN COMPUTACIÓN

EDGAR FERNANDO CARVAJAL ULLOA,

WASHINGTON JAMIL VÉLEZ NAVARRETE

GUAYAQUIL – ECUADOR

AÑO: 2017

AGRADECIMIENTO

A esta magnífica institución, ESPOL, que me ha enseñado el valor del esfuerzo y responsabilidad, y que nada es fácil en esta vida.

A mis amigos, compañeros y profesores que juntos forman parte de esta gran familia politécnica de la cual me siento orgulloso formar parte.

Edgar Fernando Carvajal Ulloa

AGRADECIMIENTO

A toda mi familia por haberme brindado su apoyo y aliento a pesar de la distancia.

A la institución que con todos los desafíos que me ha presentado, ha hecho que me esfuerce para superarlos.

A mis profesores que con sus enseñanzas han ayudado a desarrollarme profesionalmente.

A mis amigos, compañeros por formar parte de esta familia politécnica.

Washington Vélez Navarrete

DEDICATORIA

A mis padres Rubén Carvajal y Yolanda Ulloa, a mi hermano Rubén Andrés Carvajal quien ha sido más que un ejemplo para mí en cuanto a esfuerzo y perseverancia,

A mi novia Stephany Aguilar que ha sido mi apoyo en esta etapa de formación y también está terminando sus estudios universitarios.

Y, a todas las personas que gracias a su apoyo y enseñanzas brindadas han hecho posible la culminación de esta maravillosa etapa de mi vida.

Edgar Fernando Carvajal Ulloa

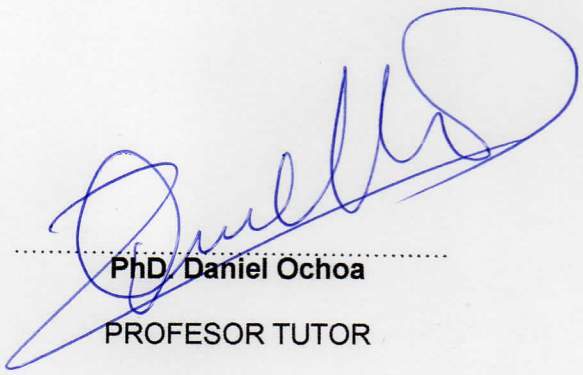
DEDICATORIA

A mis padres Washington Vélez y Lucía Navarrete que han estado conmigo en todo momento, quienes son mi mayor motivación.

A todas las personas que me han apoyado directa e indirectamente a lo largo de mi carrera universitaria.

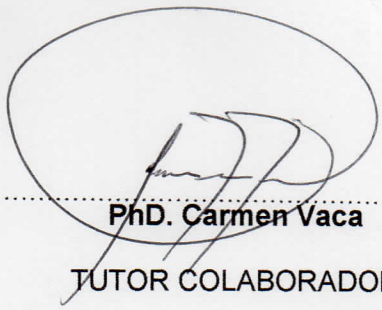
Washington Vélez Navarrete

TRIBUNAL DE EVALUACIÓN



PhD. Daniel Ochoa

PROFESOR TUTOR



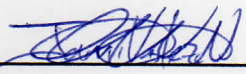
PhD. Carmen Vaca

TUTOR COLABORADOR

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, nos corresponde exclusivamente; y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Edgar Fernando Carvajal Ulloa



Washington Jamil Vélez Navarrete

RESUMEN

En las ciencias de la computación, los sistemas de minería de datos han tomado mucha relevancia en los últimos años, es por esto por lo que alrededor del mundo se evidencia un incremento significativo en cuanto a investigación y publicaciones académicas en esta área

Los resultados de estos procesos derivan en la obtención de una gran cantidad de información provenientes del área investigada.

Sin embargo, aunque estos datos tienen un gran valor cualitativo, aún se necesita de expertos en el área de computación para la interpretación o extracción de resultados de los mismos, complicando el acceso a estas herramientas de personas ajenas a las ciencias computacionales.

El objetivo de nuestro proyecto es implementar una herramienta para obtener la relación de la evolución de tópicos detectados en unidades de tiempo, esto mediante una interfaz amigable para usuarios ajenos al área informática que les permita ejecutar y posteriormente entender los resultados obtenidos de la misma.

ÍNDICE GENERAL

CAPÍTULO 1

1. INTRODUCCIÓN

1.1. Antecedentes

Las redes sociales se han convertido en plataformas de intercambio de información por su fácil uso y acceso de los usuarios. El popular servicio de microblogging Twitter, ha llegado a ser un canal influyente para la difusión de noticias llegando a generar en el 2012 alrededor de 340 millones de tweets diarios [1].

La diversidad de la información compartida en Twitter hace que su rol como fuente de noticias sea cada vez más importante para las agencias de noticias [1], pero esto a su vez genera una gran cantidad de información que sólo se publica mas no se analiza a fondo para ver su relación entre tópicos entre periodos de tiempo. Por otro lado, esta gran cantidad de información causa sobrecarga de información: un periodista no puede, por ejemplo, hacer el seguimiento de las noticias.

1.2. Descripción del problema

Existe la necesidad de analizar y extraer información relevante de gran cantidad de datos publicados en Twitter.

Cuando la información se publica en las redes sociales, en nuestro caso particular Twitter, tenemos la posibilidad de extraer dicho contenido, almacenarlo y posteriormente analizarlo.

El análisis de esta información se ha estudiado desde algún tiempo atrás por lo que existen varios artículos de investigación y propuestas científicas sobre cómo visualizar y validar la misma [2], que, si bien han logrado varios avances, no han sido de gran ayuda y uso para profesionales de la comunicación, analistas de comunicación o lectores comunes, debido a su complejidad de lectura y entendimiento.

1.3. Definiciones

En el análisis de la información se usan los siguientes términos.

1.3.1. **Tópico,** En lingüística, es una sentencia gramatical que resume en pocos términos (palabras) de qué se está hablando.

1.3.2. **Término,** Es una palabra, parte de una palabra o conjunto de palabras que forman parte de un diccionario en un lenguaje gramatical.

1.3.3. **Documento,** Conjunto de términos que refieren a un tópico o de una combinación de tópicos.

1.3.4. **Matriz documento-término,** Representación matemática en forma de matriz que describe la relación de un conjunto de términos con distintos documentos.

1.3.5. **Corpus,** Conjunto de gran cantidad documentos.

1.4. Objetivos

1.4.1. Objetivo general

Visualizar la evolución temporal de tópicos extraídos a partir de un conjunto de tweets.

1.4.2. Objetivos específicos

Implementar una interfaz para visualizar la evolución de tópicos entre dos unidades de tiempo contiguas.

Implementar la visualización para la salida de la ejecución del algoritmo JPP.

Visualizar la división y mezcla de los tópicos entre dos unidades de tiempo contiguas.

CAPÍTULO 2

2. DISEÑO DE LA SOLUCIÓN TÉCNICA A IMPLEMENTAR

Tomando en consideración el problema a resolver, detallamos a continuación nuestra solución técnica, así como los aspectos de desarrollo.

2.1. Descripción del proyecto

El algoritmo Joint Past Present (JPP) propuesto en [2] permite generar una matriz de mapeo de la evolución entre tópicos para unidades de tiempo contiguas. Dicho algoritmo recibe como entrada las matrices documento - término generadas a partir de los corpus de cada unidad de tiempo a analizar. El algoritmo detecta los tópicos para las unidades de tiempo contiguas y relaciona los tópicos generados entre estas.

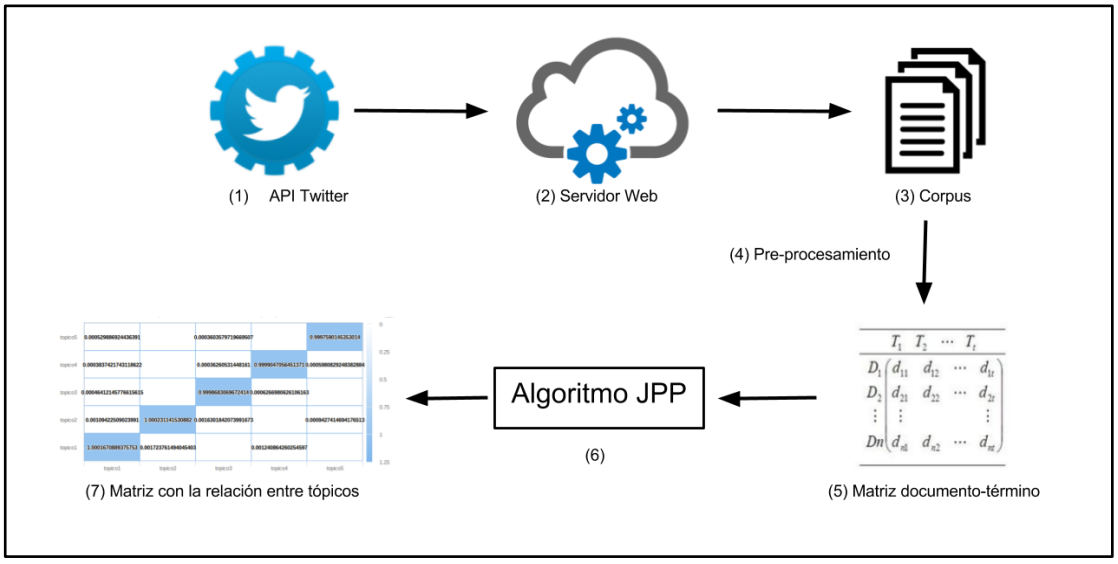


Figura 2.1. Modelo básico del sistema.

El resultado de la ejecución del algoritmo es una matriz cuadrada $M(k\ x\ k)$, donde k es el número de tópicos que ha descubierto el algoritmo del conjunto de documentos por cada unidad de tiempo analizada.

Esta matriz M, sirve para representar mediante un gráfico la evolución de los tópicos entre dos periodos contiguos. La evolución de tópicos se puede clasificar como [2]:

- Involucramiento: Un tópico del periodo inicial mantiene relación con otro tópico del periodo final
- División: Un tópico del periodo inicial se divide en otros tópicos del periodo final.
- Unión: Dos o más tópicos del periodo inicial han convergido en un tópico del periodo final.

En este proyecto el corpus de texto a analizar es un conjunto de tweets generados en Ecuador. Para extraer dicho corpus se usa el API de Twitter para extraer tweets por día, estos son almacenados en documentos, el conjunto de varios documentos forma un corpus. Con este corpus se realiza un preprocesamiento y extracción de la matriz documento - término, esta matriz es la entrada para el algoritmo JPP, el mismo que se detalla en la sección **2.2.3.1**.

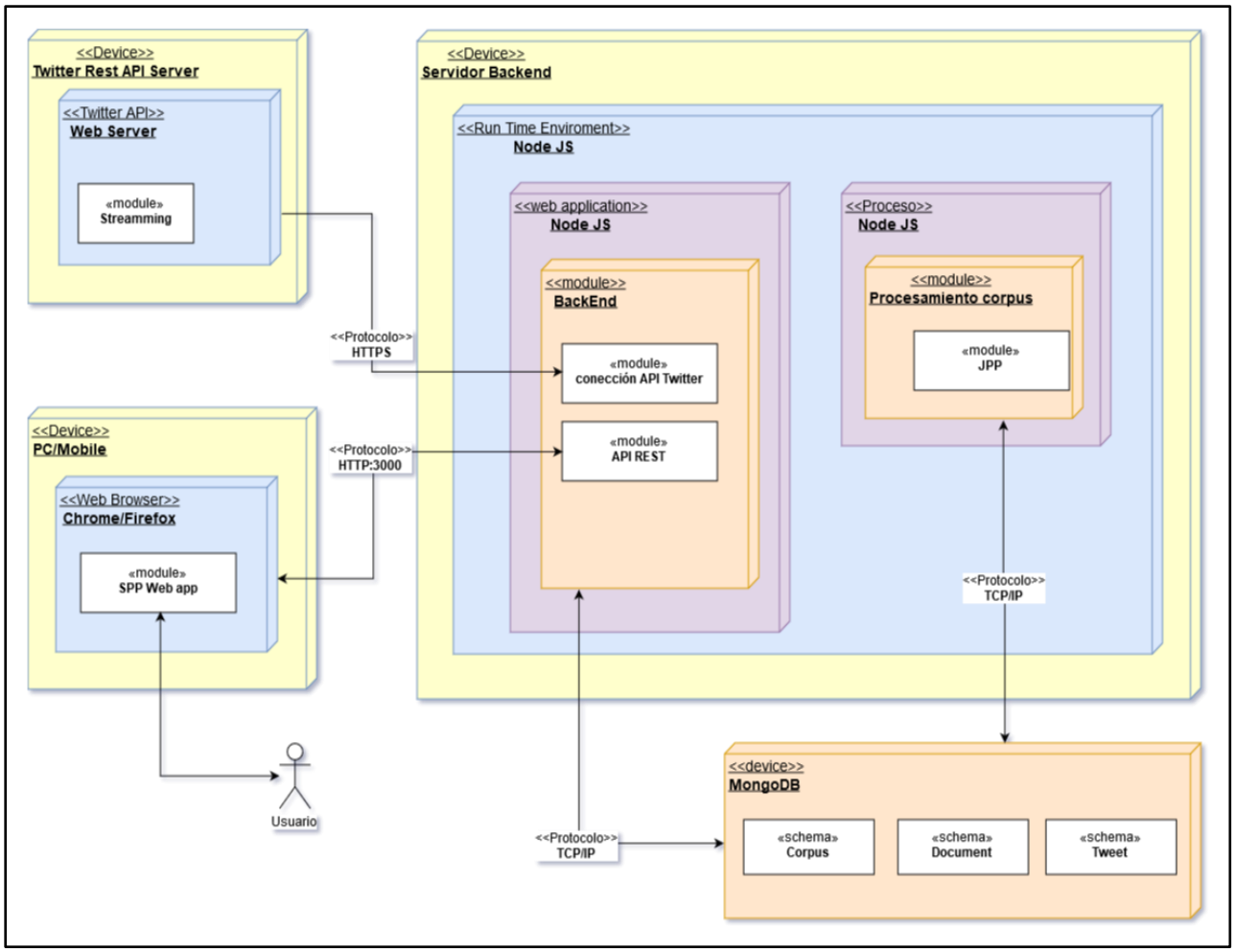


Figura 2.2. Diagrama de despliegue del sistema.

2.2. Diseño de la solución

Nuestra propuesta de solución, como se muestra en la **Figura 2.2**, es una aplicación web SPP (single page application) que será accedida por el usuario final a través de un navegador web, la cual cuenta con 4 módulos independientes.

2.2.1. Dispositivo de usuario móvil o de escritorio

El módulo de usuario final, el cual accede al aplicativo web desde cualquier dispositivo, móvil o PC de escritorio (independiente del sistema operativo), debe contar con un navegador web actualizado con soporte estándar HTML5, se recomienda el uso de Firefox, Google Chrome u Opera Browser.

En el browser, el usuario podrá realizará las siguientes actividades:

- Visualizar el streaming de tweets recopilados en tiempo real.
- Listar, seleccionar y modificar los parámetros para el cálculo del JPP de los corpus almacenados.
- Visualizar a través de una matriz de calor la relación de tópicos entre dos días.

La aplicación estará diseñada usando AngularJS, un framework web usado para diseñar apps en tiempo real y con amplia interacción cliente-servidor. La misma utiliza un API REST que se encuentra en el servidor backend descrito más adelante.

2.2.2. Twitter REST API Server

Para obtener la información a procesar, nuestra solución incluye el uso de herramientas de desarrollo que proporciona Twitter, este es un API REST con el cual obtenemos información en tiempo real de tweets y tópicos que publican diariamente los ecuatorianos.

Para acceder a la funcionalidad del API de Twitter, se necesita una conexión HTTPS a sus endpoints públicos establecidos en la documentación provista por Twitter.

Esta información una vez obtenida, se procesa en el servidor backend para posteriormente ser almacenada y utilizada como entrada para el algoritmo JPP.

El procesamiento se describe más adelante en la sección **2.2.3**

2.2.3. Servidor Backend

El servidor Backend es la instancia encargada de conectar, procesar y almacenar la información de los diferentes módulos del proyecto, es decir, la aplicación web, el API REST Twitter y los diferentes procesos (SO) que, para el procesamiento de la información, se creen en el mismo equipo.

2.2.3.1. Aplicación web NodeJS

La aplicación web está implementada en NodeJS, un entorno en tiempo de ejecución usado en la creación de API's REST.

Nuestro servidor cuenta con dos módulos: el del API REST que será utilizada por la aplicación web en AngularJS y el módulo para el procesamiento de la información obtenida del API de Twitter.

2.2.3.2. Algoritmo JPP

El algoritmo utiliza NMF (factorización matricial no negativa) para descomponer la matriz documento-término en dos matrices: matriz documento-tópico y matriz tópico-término [2].

La matriz documento-tópico es una matriz que representa la probabilidad de que un documento pertenezca a un tópico.

La matriz tópico-término es una matriz que representa la probabilidad de que un término pertenezca a un tópico.

El algoritmo asume que, para dos periodos distintos de tiempo, cada matriz documento - termino $X^{(t)}$se descompone de la siguiente manera:

$$X^{(t)} \approx W^{(t)} H^{(t)} \quad (2,1)$$

$$X^{(t)} \approx W^{(t)} M^{(t)} H^{(t-1)} \quad (2, 2)$$

Donde la matriz $W^{(t)}$representa una matriz documentos-tópicos, y $H^{(t)}$ una matriz tópicos-términos.

Para obtener la matriz $M^{(t)}$ que obtiene el mapeo de la relación entre tópicos del día A con el día B, el algoritmo se aplica dos veces:

- la primera vez, solo recibe como entrada la matriz $X^{(t)}$ que es la matriz documento-término del día A de esta ejecución la matriz resultante $H^{(t)}$ se usa como entrada para la segunda ejecución del algoritmo.
- la segunda vez, recibe la matriz $X^{(t)}$ del día B y la matriz $H^{(t)}$ que es la matriz tópicos-términos del día A

De la salida de la última ejecución del algoritmo, la matriz $M^{(t)}$ es la que contiene la matriz Mde mapeo de la evolución de los k tópicos del día A con los k del día B.

2.2.4. Base de datos

La base de datos se incorporó al sistema debido a la necesidad de almacenar información de una manera consistente y con el objetivo de procesar la misma una vez almacenada,

Para la misma, se eligió MongoDB una base de datos no relacional en la cual se almacena la información como en la **Figura 2.3**.

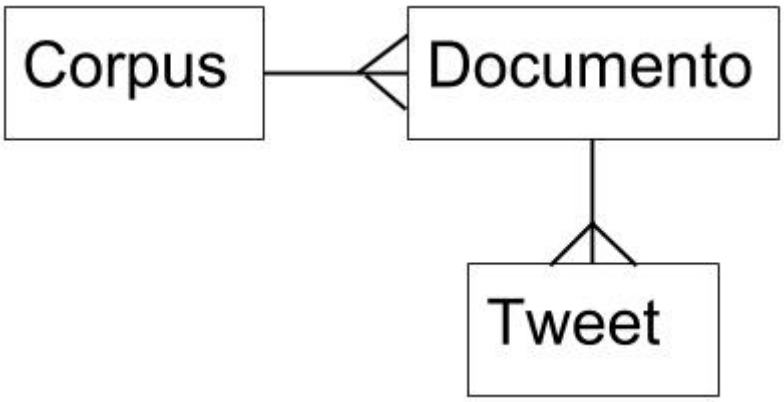


Figura 2.3. Esquema de la base de datos.

Cada día de stream se representa como un **corpus**, cada corpus tiene 96 **documentos**, esto es, uno cuarto de hora de streaming; y cada

documento tendrá una cantidad **indefinida de tweets** que serán recopilados en ese lapso.

Actualmente no se cuenta con un servidor físico de base de datos. En su lugar, se utiliza una versión gratuita de desarrollo en la nube, obtenido en el sitio web oficial de MongoDB (1).

2.3. Justificación de herramientas

2.3.1. MongoDB

Para el almacenamiento de información requerimos de un sistema que nos ofrezca consistencia y posibilidad de realizar búsquedas de la misma. Por tal motivo un sistema de archivos fue descartado.

Lo cual nos dejó como elección las bases de datos relacionales y las no relacionales, estas últimas con la propiedad de no requerir un esquema complejo para el almacenamiento de la información, si no, que basan su estructura mediante documentos de cualquier tamaño.

Pero la métrica más importante que usamos para la elección de la misma es la definida por el teorema CAP, el cual dice que ninguna base de datos puede poseer alta consistencia, disponibilidad y capacidad alta de particionamiento al mismo tiempo.

Apache CouchDB y MongoDB terminaron siendo nuestras mejores opciones en cuanto a nuestros requerimientos.

Selección: MongoDB fue nuestra elección, debido a su capacidad de manejar información que puede no estar sujeta siempre a esquemas, su alto potencial en **particionamiento** lo cual resulta finalmente en alta escalabilidad, muy necesaria debido a la enorme cantidad de almacenamiento de datos del proyecto y, también por su propiedad de alta **disponibilidad,** esto es, que siempre se obtendrá información cuando lo requiera aun cuando esta no posea alta consistencia.

Finalmente, otro parámetro que usamos para la elección de Mongo fue su amplia documentación y una gran comunidad que hace uso de la

misma, las cuales son importantes en el desarrollo de proyectos con limitada cantidad de tiempo para su implementación.

2.3.2. NodeJS

Para la conexión de nuestros módulos requerimos de un servidor que cumpla con las siguientes características:

- Realizar requerimientos continuos y de manera indefinida a distintos servicios web.
- Responder a los requerimientos que haga la aplicación web.
- Permitir una conexión full dúplex en tiempo real con cada cliente que use el aplicativo web.
- Ejecución de los algoritmos de procesamiento y cálculo implementados para el proyecto.
- Administrar y controlar las conexiones de todos los dispositivos conectados, esto es, evitar la sobrecarga de procesamiento por parte del equipo en el cual se aloja el servidor.

Para suplir todas estas necesidades requerimos de un sistema que sea capaz de responder a todos estos requerimientos y al mismo tiempo procesar complejos algoritmos matemáticos que harían uso intensivo de memoria y CPU.

Además de esto, nuestra elección estuvo sesgada por al factor tiempo que manejamos en un proyecto como este, por tal motivo decimos usar un sistema que permita implementar en un mismo lenguaje de programación lo siguiente:

- Servidor web HTTP.
- Servidor de Websockets.
- Manejo de manera asíncrona un promedio de 200 requerimientos web por minuto a un REST API externo, en nuestro caso particular, Twitter.
- Librerías para la conexión con MongoDB, y

- La capacidad de ejecutar operaciones matemáticas de alta intensidad de cómputo, sin afectar el rendimiento de los requerimientos anteriores.

Por tal motivo, teníamos la opción de usar Django/Python o NodeJS/Javascript.

Finalmente, nuestra decisión de usar **NodeJS** fue debido a la propiedad de que este es un runtime environment (ambiente en tiempo de ejecución), el cual compila el código en JavaScript directamente a lenguaje de máquina usando un compilador llamado V8, dándole una amplia ventaja sobre Python el cual es interpretado; otro punto a su favor fue que para su desarrollo usaremos el mismo lenguaje en el backend como en el frontend.

2.3.3. AngularJS y APIs Javascript

Para la aplicación web, se requiere una librería o framework que nos permita dividir el proyecto en componentes y controladores, los cuales nos permiten reutilizar código de la mejor manera posible, AngularJS proporciona dicha funcionalidad.

Además de esto, se necesitaba una aplicación que sea capaz de responder de manera inmediata a los cambios y respuestas generadas por el servidor, en un resumen de los requerimientos tenemos:

- Conexión full dúplex con el servidor backend.
- Actualizaciones y cambios en tiempo real de las vistas según los cambios en el servidor.
- Procesar ciertos cálculos matemáticos en el navegador web (sin interrumpir la navegación del usuario).

Para las conexiones full dúplex, se utiliza Websockets, una tecnología que hace posible el intercambio de información servidor-cliente sin la necesidad de que exista un requerimiento para iniciar la misma.

Para las actualizaciones en tiempo real y la creación de un SPP (Single Page Application) hicimos uso de AngularJS, un framework Javascript

desarrollado por Google que permite la conexión directa entre el DOM (HTML) y nuestra aplicación web (código en JavaScript).

Finalmente, para la ejecución de operaciones matemáticas que no interrumpan la navegabilidad del usuario, esto debido a que JavaScript es hilo único, hicimos uso de WebWorkers, una tecnología que, aunque tiene ciertas limitantes finalmente pudimos implementar algo llamado Generic WebWorkers, el cual describiremos más adelante.

CAPÍTULO 3

3. IMPLEMENTACIÓN DE LA SOLUCIÓN

La implementación del sistema se dividió en tres módulos:

- Captura de información (tweets),
- Estandarización de la información y preprocesamiento
- Ejecución del algoritmo JPP sobre la información estandarizada y generación de gráficas para el usuario final.

3.1. Captura de información (tweets)

Para la captura de información se utiliza el API REST de Twitter al cuál se accede por medio del servidor backend en el entorno NodeJS y la información capturada se almacena en la base de datos no relacional MongoDB.

Para el uso del API de Twitter es necesario enviar parámetros de búsqueda, de los cuales elegimos dos: un bounding box (4 coordenadas geográficas) que representan un área geográfica del Ecuador y una lista de track Ids (cuentas de usuarios a las cuales se capturan sus tweets), los cuales son cuentas de Twitter con importante influencia en las redes sociales de acuerdo con 4 tópicos centrales: deportes, política, noticias y temas de actualidad. Entre estos, por ejemplo, están todas las cuentas de Twitter de los medios de comunicación nacionales. **Figura 3.1**.

```
parametros { locations: '-80.189855,-3.309067,-77.645968,0.446412',
  follow: '161775175,56039780,164715111,175114792,112757144,355809231,59
714229,15447575,419971210,198666634,176836959,1543728618,174825461,39022
0919,2409683900,395429154,468586302,1601592949,189124591,195048109,57380
8532,185658756,300390462,24776620,315377387,220332032,197932909,12893796
7,285137536,271975968,202792190,216783643,56502954,215996236,96789803,82
25692,133184048,97513349,95217117,3174317410,3169785371,51254405,2567476
96,35047698,35289042,14333756,175447500,202779862,183661586,244136790,20
3093566,202782111,812220439' }
Inicio de streamming de tweets
```

Figura 3.1. Parámetros de búsqueda.

Posteriormente el servidor web, a medida que los tweets son capturados en modalidad streaming de Twitter, elimina información irrelevante de cada tweet, como por ejemplo caracteres especiales, emoticones, enlaces, entre otros.

Finalmente se almacena cada tweet en una base de datos con una clave foránea de su respectivo documento y corpus correspondiente. Un día de captura de tweets se denomina **Corpus**, y contiene 4 documentos por cada hora en los cuales se registra un total de 40 tweets en promedio por cada documento.

3.2. Estandarización de la información y preprocesamiento

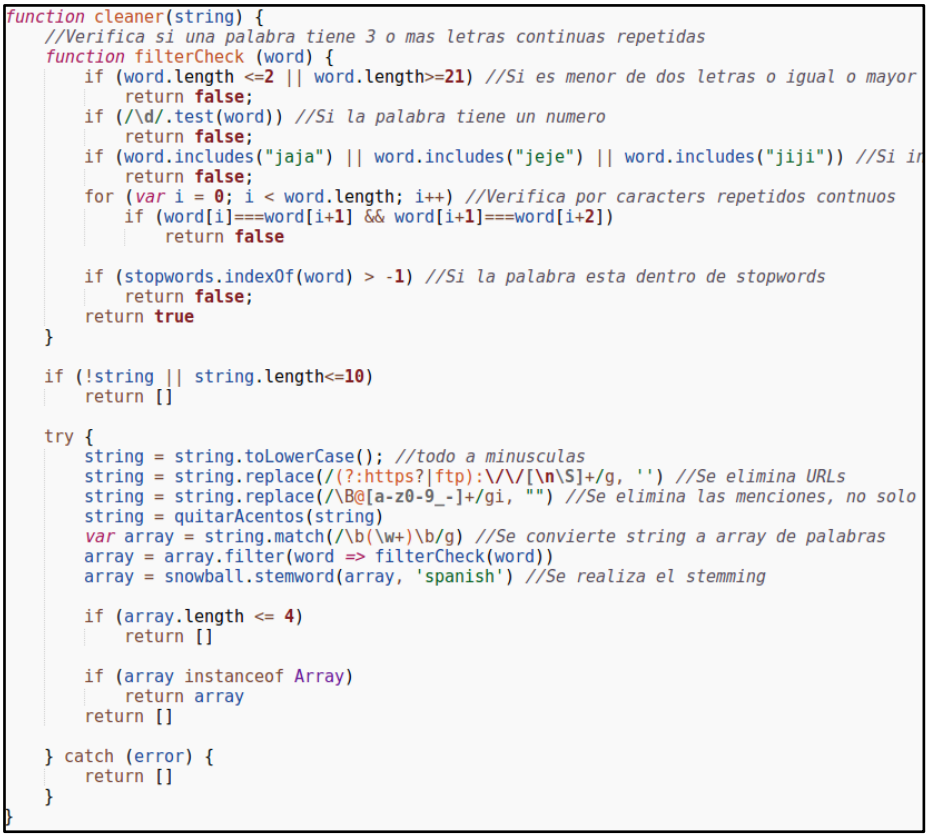
Con todos los corpus almacenados en la base de datos, necesitamos procesarlos para generar por cada uno de estos, una matriz $X(n\,xm)$, que es la entrada del algoritmo JPP, donde n es el número de documentos y m el número de términos que existen en el corpus.

Los valores almacenados en la celda (i, j) de la matriz $X(n\,xm)$ representan el valor generado por el cálculo del TF-IDF del término j en el documento i de ese corpus.

Para realizar este preprocesamiento se realizó el siguiente procedimiento:

3.2.1. Tweet a Array[string]

A cada tweet se le aplicó una función de limpieza llamada cleaner, la cual elimina términos irrelevantes, acentos, realiza steaming a las palabras, etc. **Figura 3.2**. Como resultado final de esta función obtenemos un Array de strings el cual será usado posteriormente.



```
function cleaner(string) {
    //Verifica si una palabra tiene 3 o mas letras continuas repetidas
    function filterCheck (word) {
        if (word.length <=2 || word.length>=21) //Si es menor de dos letras o igual o mayor
            return false;
        if (/\d/.test(word)) //Si la palabra tiene un numero
            return false;
        if (word.includes("jaja") || word.includes("jeje") || word.includes("jiji")) //Si ir
            return false;
        for (var i = 0; i < word.length; i++) //Verifica por caracters repetidos contnuos
            if (word[i]===word[i+1] && word[i+1]===word[i+2])
                return false

        if (stopwords.indexOf(word) > -1) //Si la palabra esta dentro de stopwords
            return false;
        return true
    }

    if (!string || string.length<=10)
        return []

    try {
        string = string.toLowerCase(); //todo a minusculas
        string = string.replace(/(?:https?|ftp):\/\/[\n\S]+/g, '') //Se elimina URLs
        string = string.replace(/\B@[a-z0-9_-]+/gi, "") //Se elimina las menciones, no solo
        string = quitarAcentos(string)
        var array = string.match(/\b(\w+)\b/g) //Se convierte string a array de palabras
        array = array.filter(word => filterCheck(word))
        array = snowball.stemword(array, 'spanish') //Se realiza el stemming

        if (array.length <= 4)
            return []

        if (array instanceof Array)
            return array
        return []

    } catch (error) {
        return []
    }
}
```

Figura 3.2. Impresión de pantalla función cleaner.

Este esquema hace posible no sólo pre-procesar información proveniente de tweets, sino que también permite grandes documentos con textos contiguos, para esto el sistema toma toda la información como si fuera un solo tweet.

3.2.2. Creación del corpus

Para la creación de un corpus con información limpia (pre-procesada), mediante pruebas se determinó que era necesaria una alta demanda de uso del CPU y memoria del sistema. Se utilizó el mecanismo de programación concurrente para que un fragmento de código se ejecute por separado y no afecte en lo absoluto al hilo principal del programa donde funciona la recolección de tweets y el servidor web. **Figura 3.3.**

Finalmente, este procedimiento nos da como resultado un objeto con dos propiedades: un conjunto de palabras únicas que existen en un corpus y una lista de diccionarios, esta lista tiene un diccionario por cada documento, y cada diccionario tiene como clave cada palabra de

dicho documento y como valor tiene la frecuencia de cada palabra en el mismo.

```
const { cleaner } = require("../util/process.js")

process.on('message', function (documentos) {
    var corpus = [] // [Map] new Map("word", frecuency)
    var setPalabras = new Set() //Palabras sin repetir del corpus

    documentos.forEach(doc => {
        var map = {}
        var docWords = doc.tweets.reduce((words, t) => [...words, ...cleaner(t) ], [])

        docWords.forEach(word => {
            if (word in map)
                map[word]++
            else
                map[word] = 1
        })

        for (var key in map)
            setPalabras.add(key);

        corpus.push(map)
    })
    setPalabras = [...setPalabras]
    process.send({setPalabras, corpus})
})
```

Figura 3.3. Código de generación del corpus.

3.2.3. Generación de la matriz documentos términos (TF-IDF)

La matriz $X(n\ xm)$ de un corpus es la información que necesita nuestro algoritmo para procesar y posteriormente poder visualizar la información. La generación de la misma también requiere un alto uso del CPU y memoria del sistema, por lo que fue necesario de igual manera que en el punto **3.2.2** utilizar el mecanismo de programación concurrente para que una sección de código se ejecute en proceso separado del hilo principal donde se ejecuta la aplicación web. **Figura 3.4**.

```
process.on('message', function (data) {
    var X = data.setPalabras.reduce((arr, word) => [...arr, tf_idf(data.corpus, word)], [])
    process.send(X)

})

process.on('uncaughtException', function (err) {
    console.log("err", err)
    process.send({error: err})
})


// corpus = [{word: frecuency}]
function tf_idf (corpus, word) {

    var tf = (doc, word) => Math.log(1 + (doc[word]|| 0) )
    var nt = corpus.reduce((nt, doc)=> doc[word] ? ++nt : nt ,1)
    var idf = Math.log(1 + (corpus.length / nt))

    return corpus.reduce((xT, doc) => [...xT, tf(doc, word)*idf], [])
}
```

Figura 3.4. Código de generación de matriz X.

3.3. Generación de gráficas para el usuario final.

Una vez que cada corpus ha sido procesado se tiene como resultado un set de palabras y su respectiva matriz $X(n\,xm)$. Cuando el usuario elija dos días para visualizar la relación entre sus tópicos, el sistema obtiene la $X(n\,xm)$ de cada día y envía ambas matrices como entrada al algoritmo JPP. El algoritmo JPP, con esta información de entrada, genera la gráfica que contiene una matriz de la relación de los tópicos detectados por estas dos unidades de tiempo seleccionadas por el usuario.

La visualización presentada al usuario consiste en tres fases las cuales son:

3.3.1. Visualización y selección de corpus (usuario final)

EL usuario final mediante el aplicativo web puede ver la lista de todos los corpus de información que han sido almacenados y pre-procesados hasta el momento como se muestra en la **Figura 3.5**.

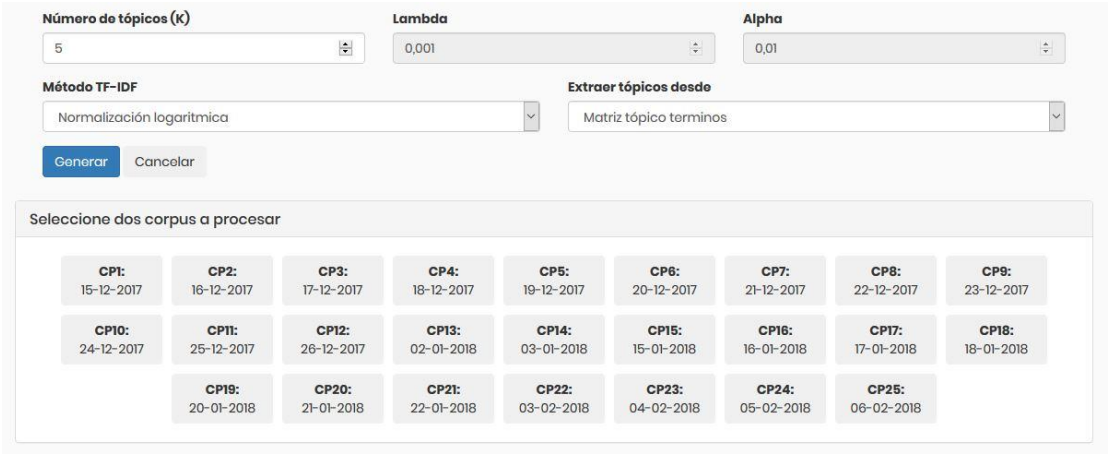


Figura 3.5. Corpus pre-procesados en la aplicación web.

A continuación, debe seleccionar dos de ellos, elegir un valor para el parámetro k (número de tópicos a detectar para cada unidad de tiempo) y lambda (parámetro de nivelación) y escoger el método para el cálculo del TF-IDF (utilizado en el preprocesamiento del corpus) y el método para la extracción de tópicos. Finalmente, al hacer clic en el botón generar, el usuario visualizará una pantalla de espera mientras el sistema procesa esta información de entrada.

El aplicativo web a continuación creará una conexión full dúplex usando la tecnología Web socket mediante la cual se informará en tiempo real al usuario el momento en que su resultado esté listo, también si ocurrió un error en el procesamiento.

3.3.2. Selección de corpus provenientes de archivos.

Si bien el aplicativo fue desarrollado con la finalidad de análisis y procesamiento de datos extraídos de redes sociales, también se desarrolló una sección en la que el usuario podría analizar corpus provenientes de archivos JSON.

Para lo cual, se deberían de seguir los siguientes pasos.

- **Ingresar dos corpus en el formato indicado (ver figura 3.6)**
  En esta sección el usuario tendrá libre elección sobre los parámetros de procesamiento, esto es: número de tópicos, lambda, alpha, método TF-IDF, desde donde se extraen los

tópicos y el lenguaje de stemming. Finalmente ingresa los archivos JSON y el proceso continuará en la sección 3.3.3.

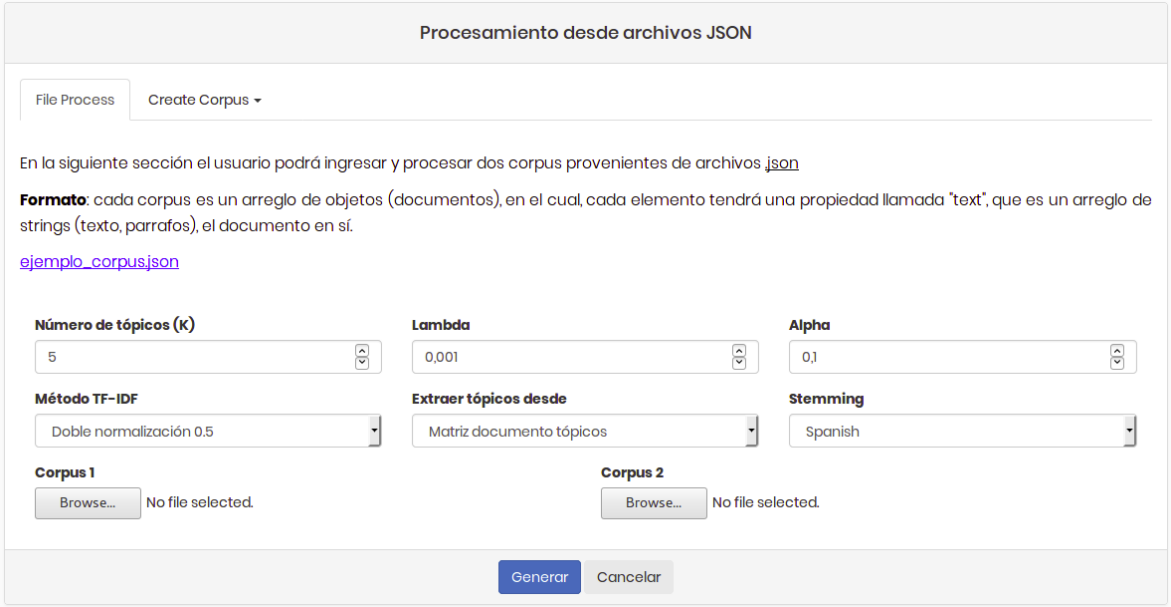


Figura 3.6. Selección de corpus mediante archivos.

- **Generar archivos JSON con formato adecuado desde aplicativo.**

  En caso de que el usuario no posea los archivos indicados, o no tengan el formato correcto, podrá generar los mismos desde la sección “Create Corpus”. Con dos opciones:

  “**From Text**”, en la cual el usuario ingresará tantos textos como documentos quisiera que tenga su corpus (copiar-pegar) ver **Figura 3.7**.

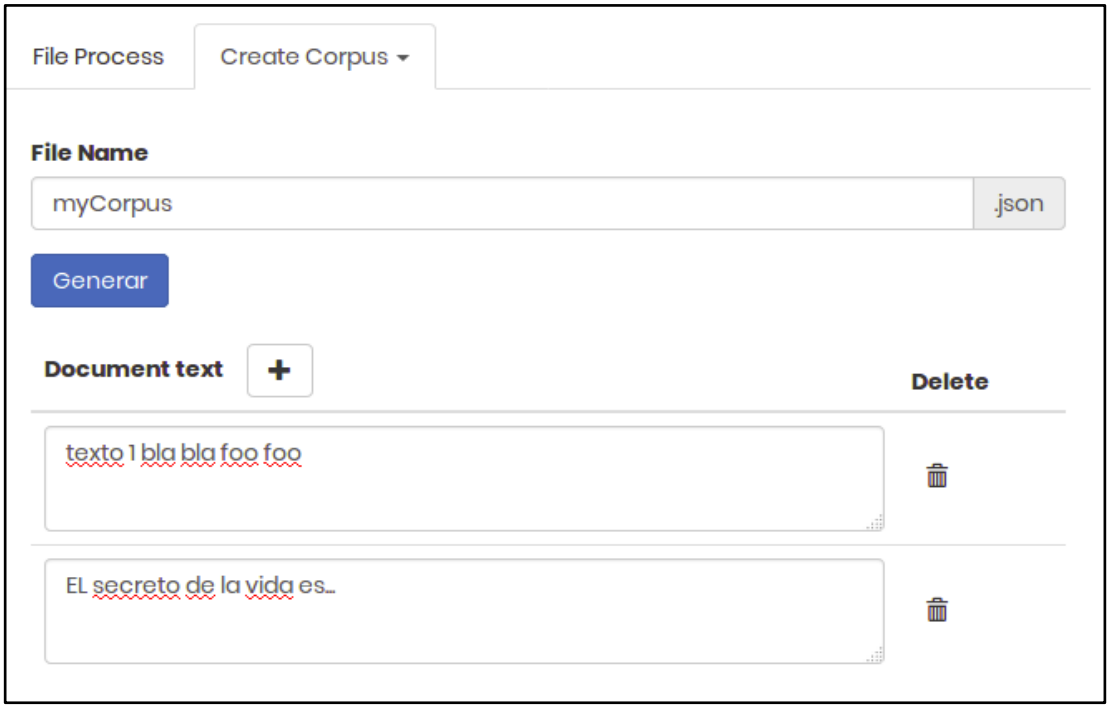


Figura 3.7. Creación de corpus mediante texto.

"**From files**", EL usuario seleccionará tantos archivos como documentos tendrá el corpus. Los archivos serán leídos en su totalidad como texto, no importa el formato que estos tengan, ver **Figura 3.8**.

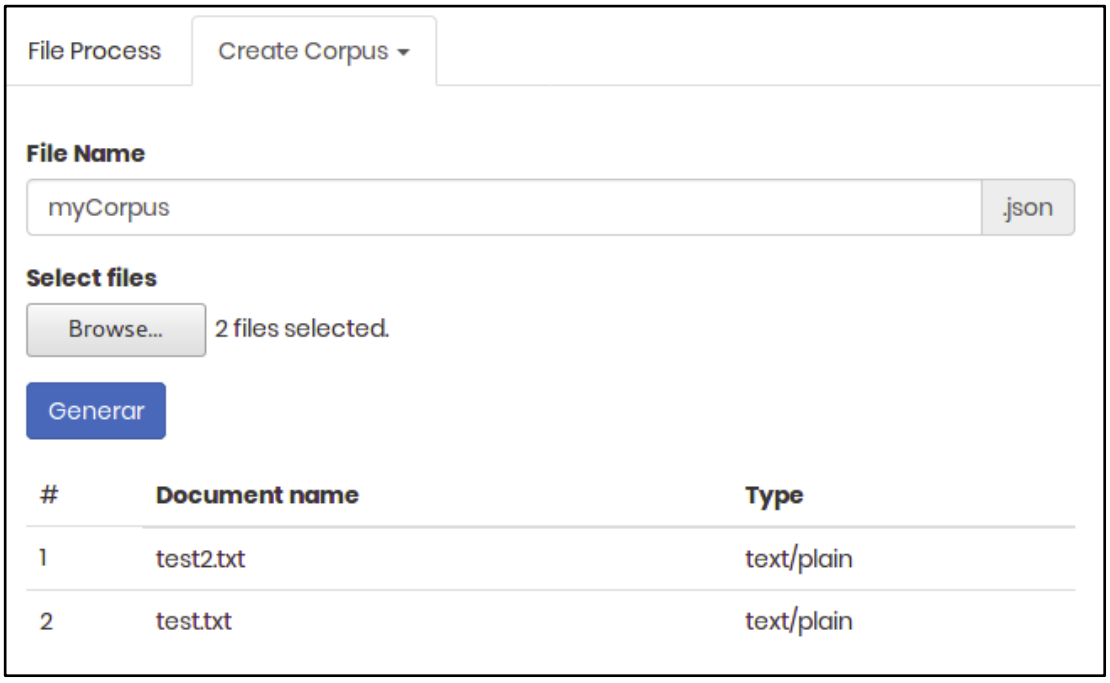


Figura 3.8. Creación de corpus mediante archivos.

3.3.3. Procesamiento de los corpus

En el servidor se inicia el procesamiento de la información seleccionada por el usuario, para lo cual de igual manera que en ocasiones anteriores donde se requería un elevado uso del CPU y memoria del sistema, se procedió a utilizar programación concurrente para el segmento de código encargado del algoritmo JPP.

La información resultante de este procesamiento es la matriz $M(k\ x\ k)$ donde k es el número de tópicos que el usuario escogió; y, además un arreglo de las palabras de cada tópico por cada uno corpus los corpus, es decir, dos arreglos de arreglos de palabras.

3.3.4. Visualización de resultados

Terminada la etapa de procesamiento, se hace un broadcast a todos los usuarios que han solicitado un requerimiento similar, esto es, se envía la matriz M junto a sus arreglos de tópicos.

En el aplicativo web se creó una función asíncrona llamada **requestJPP** ver **Figura 3.9** para abstraer las funcionalidades y métodos que cuentan los sockets, estos son eventos, listeners, callbacks y demás.

Una vez se obtuvo la respuesta del procesamiento, se direcciona al usuario a la pantalla de la visualización de la relación entre los tópicos detectados para los días indicados por el mismo.

Esta pantalla tiene un mapa de calor, y la lista de tópicos detectados por cada unidad de tiempo previamente seleccionada ver **figura 3.10**.

En la lista de tópicos de cada unidad de tiempo, el usuario puede hacer clic en el botón del índice de uno de ellos y visualizará un gráfico de barras que representa la composición de dicho tópico de una unidad de tiempo con respecto a los tópicos de la otra unidad de tiempo ver **figura 3.11**.

```
var socket = io.connect('http://localhost:3001', {'forceNew':true })

socket.on("jpp", res => {
    var event = new CustomEvent(res.peticion, {detail: res})
    document.dispatchEvent(event)
})

socket.on("disconnect", ()=> {
    document.dispatchEvent(new Event("disconnect"))
})

//Key, es el id de suscripcion
//data, información a enviador
function requestJPP (peticion, data) {
    return new Promise((resolve, reject)=> {

        if (socket.disconnected)
            return reject("Server disconnected")

        function response (e) {
            var res = e.detail
            if (res.error)
                return reject(res.error)
            document.removeEventListener(peticion, response)
            document.removeEventListener("disconnect", disconnect)
            resolve(res)
        }

        function disconnect () {
            document.removeEventListener(peticion, response)
            document.removeEventListener("disconnect", disconnect)
            reject("Server disconnected")
        }

        socket.emit("jpp", {peticion, data})
        document.addEventListener(peticion, response, false)
        document.addEventListener("disconnect", disconnect, false)
    })
}
```

Figura 3.9. Abstracción del uso de Websockets.

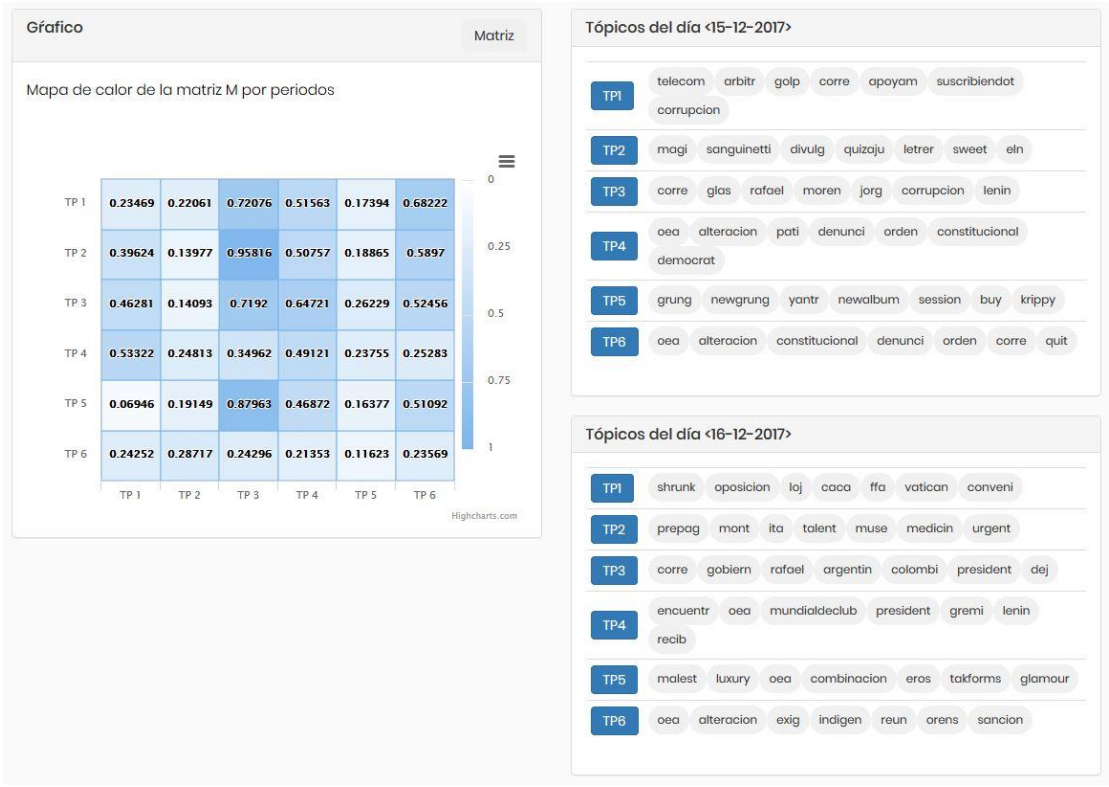


Figura 3.10. Visualización de matriz de relación de tópicos con las listas de tópicos para las dos unidades de tiempo procesadas. Para el ejemplo mostrado el tópico 3 del día 1 y el tópico 2 del día 2 tienen una relación de 0.958116.

Figura 3.11. Visualización de evolución de un tópico en particular. Para el ejemplo mostrado el tópico 3 del día 1 y el tópico 2 del día 2 tienen una relación de 0.958116.

CAPÍTULO 4

4. RESULTADOS DE LA IMPLEMENTACIÓN

Para el desarrollo de cada uno de estos módulos tuvimos que resolver 3 grandes problemas para su implementación:

4.1. Hacer uso de todos los recursos disponibles en el equipo sin sobrecargarlo

Debido a la intensidad de procesamiento que requiere este sistema, existen momentos en que se requiere procesar varios lotes de información.

Y, aunque utilizamos procesos separados para ejecutar operaciones de alto uso del CPU, tampoco podíamos crear procesos sin límite alguno.

```
//Concurrency == 1, serie
//Concurrency == Infinity, todo en paralelo
function each(array, fn, concurrency) {
    return new Promise((resolve, reject) => {
        concurrency = (concurrency > 0) ? concurrency : 1
        var used = 0
        var size = array.length
        var resolved_data = []

        while (array.length > 0 && used < concurrency) {
            fn(array.pop(), next, error), used++
        }

        function next(data) {
            resolved_data.push( data )
            used--
            if (resolved_data.length == size)
                return resolve(resolved_data)
            if (used >= concurrency || array.length == 0)
                return
            used++
            fn(array.pop(), next, error)
        }

        function error(error) {
            used = Infinity
            return reject(error)
        }
    })
}
```

Figura 4.1. Función para despachar tareas según grado de concurrencia.

Una vez ingresados estos parámetros, se ejecutarán en paralelo un número de elementos igual al número de cores, cuando un resultado sea obtenido automáticamente se despachará otro elemento, manteniendo así siempre ocupados los cores del sistema sin sobrecargarlos. **Figura 4.1**.

4.2. Necesidad de abstraer el procedimiento de ejecutar secciones de código de forma concurrente

Debido a la propiedad de NodeJS con JavaScript de ser single thread para la ejecución del runtime environment, se separó en varias ocasiones el procesamiento de secciones de código usando procesos hijos del programa principal.

El problema surgió en la administración de todos estos procesos con tiempo de vida no estimado, es decir se creaban para la ejecución de un código, pero una vez obtenido el resultado había que cerrar el mismo o retornar sus errores.

Para solucionar esto, creamos una función genérica asíncrona la cual recibe como parámetros el nombre del programa a ejecutar de manera concurrente e información que le quiera enviar al mismo, finalmente una vez obtenido el resultado, simplemente es retornado y esta se encargará de su tiempo de vida. **Figura 4.2**.

```
function processPromise (path, data) {
    return new Promise(function(resolve, reject){
        var child = fork(path)

        child.send(data)

        child.on("message", function (result){

            if (result.error) {
                reject(new Error(result.error))
                return child.kill()
            }

            resolve(result)
            return child.kill()
        })
    })
}
```

Figura 4.2: Función que ejecuta de manera concurrente un programa enviado como parámetro.

4.3. Procesamiento de operaciones matemáticas en el Navegador Web.

Para distribuir la carga de procesamiento y gracias a que el lenguaje de programación usado en el servidor es el mismo que usamos en el lado del cliente, esto es JavaScript; vimos la oportunidad de ejecutar ciertas operaciones en el navegador.

El problema se evidenció ya que cualquier operación de alto grado de cómputo bloquea todo script ejecutándose, congelando la página web hasta que termine su procesamiento.

Para solucionar esto existe un API JavaScript llamado WebWorkers (WW) en el que se puede ejecutar un código predefinido fuera del hilo principal, el problema es:

- No es genérico, es decir debe de existir un archivo predefinido.
- No acepta la transferencia de funciones (reusar código)
- Requiere el manejo de varios eventos provenientes del WW hacia la página principal.

Como un proyecto personal, Edgar Carvajal había estado trabajando en una librería JavaScript llamada GenericWebWorker [3] la cual permite abstraer todo lo mencionado y ejecutar una función callback en un webworker genérico.

```
var worker = new GenericWebWorker()

worker.exec(function() {

    //This will block the code for a long time, but it is n
    var a = 0
    for (var i = 0; i < 100000000000; i++) //blocking code
        a += i

    return a
})
.then(res => console.log(res)) //Getitng the result back in
.catch(()=>{})
```

Figura 4.3 Sección de código ejecutado fuera del hilo principal.

Por tal motivo se concluyó el desarrollo de la misma e incorporó a este proyecto. **Figura 4.3**.

4.4. Ruido en información capturada (tweets).

Los datos provenientes de redes sociales traen consigo una enorme cantidad de información irrelevante (ruido) que si no es eliminada correctamente puede derivar en resultados totalmente distintos a los esperados.

Inicialmente en el preprocesamiento de la información el sistema calculaba un diccionario de datos de con promedio 60,000 palabras por cada corpus, lo cual significa una gran dispersión en la matriz X (documento - término), esto ocasionó:

- Aumento significativo en el tiempo de procesamiento por cada corpus.
- Resultados sin sentido en la visualización.
- En ciertos corpus, el algoritmo converge en sólo dos iteraciones cuando lo normal eran 100.

Para solucionar este inconveniente tuvimos que aplicar cambios significativos al aplicativo, estos son:

4.4.1. Filtro de usuarios

En la recolección de tweets, hubo la necesidad de filtrar aquellos provenientes de usuarios válidos, esto es, que cumplían con ciertos requisitos.

Para esto, analizamos el perfil del usuario y se compara el TFF Ratio (Twitter Following Follower Ratio), que es la relación entre el número de seguidores y el número de cuentas seguidas [4]. Si este cociente resultaba mayor a 1, el tweet se almacenaba. Con esto se pudo eliminar una gran cantidad de tweets generados por bots o de usuarios con poca relevancia.

4.4.2. Filtro diccionario de palabras.

Si bien en el preprocesamiento desechamos una gran cantidad de términos que no se pueden considerar palabras, tuvimos que aplicar

otros más, uno de ellos fue agregar un filtro de stopwords en idioma inglés y conseguir el set de palabras que existe en el idioma español.

Debido a que estas nuevas funcionalidades ocasionaron un aumento en promedio de 40 segundos en el tiempo de procesamiento de cada corpus, se decidió incluir estas propiedades como alternativas de elección, es decir se dejó a discreción del usuario si se procesa o no la información con estas funcionalidades.

4.4.3. Nuevo método de extracción de tópicos.

Inicialmente los tópicos de un corpus sólo se extraían de la matriz H (tópico - término), esto ocasionó que en corpus con documentos muy extensos se obtenían demasiados términos relevantes por cada tópico y, nosotros al sólo hacer uso de los primeros 7, terminamos visualizando información sin relación alguna.

Para cambiar esto se decidió agregar otro método para extraer tópicos, el cual tiene el siguiente procedimiento (**ver Figura 4.4**.)

1. Se genera la transpuesta de la matriz W (documento - tópico).
2. Por cada tópico se extrae los mejores 10 documentos del mismo.
3. Cada documento se reemplaza con un arreglo de los 7 términos con mayor frecuencia del mismo, en orden de mayor a menor.
4. Reducimos cada tópico desde (Arreglo de Arreglo te términos) a (Arreglo de términos).
5. Finalmente se elige los primeros 7 términos de cada tópico.

Esta nueva forma de extracción de tópicos se encuentra como parámetro de entrada en el inicio del procesamiento, es decir el usuario escoge cuál de los dos métodos usar.

Nuestra recomendación es usar el método (tópico término) cuando se tenga pocos documentos y estos sean bien extensos, mientras que se debería de usar (documento tópico) en corpus con muchos documentos (más de 60).

```
//De cada tópico, elige los 10 mejores documentos y extrae sus palabras con mayor frecuencia.
function extraerDocTopicos (W, corpus, Matrix) {

        function findIndicesOfMax(arr, count) {
                var new_arr = arr.map((val, index)=> { return {val, index} })
                new_arr.sort((a,b)=> b.val-a.val)
                return new_arr.slice(0, count).map(it => it.index)
        }

        var to_doc = new Matrix(W).T.data //Topico x Doc
        to_doc = to_doc.map(t => findIndicesOfMax(t, 10)) //n mejores documentos

        return to_doc.map(topico => {
                topico = topico.map(index_doc => corpus[index_doc].map)
                topico = topico.map(doc => {
                        doc = [...doc.entries()].sort((a, b)=> b[1]-a[1]) //[ ['perro', 233], ['gato', 9],
                        return doc.slice(0, 7).map(obj => obj[0]) //['perro', 'gato', 'caballo'] //Each doc
                })

                var words_topico =  []
                while (topico.some(doc => doc.length > 0)){
                        for (var doc of topico) {
                                words_topico.push( doc.shift() ) //Se agrega al arreglo el primer elemento
                        }
                }

                return [...new Set(words_topico.filter(w => w !== undefined))].slice(0,7)
        })
}
```

Figura 4.4 extracción de tópicos.

4.4.4. Nuevo TF-IDF.

Inicialmente se contaba únicamente con un método de cálculo para el TF-IDF, el cual usa normalización logarítmica para el cálculo del TF.

Este método es el responsable de la generación de matrices dispersas en corpus con demasiados términos, esto debido a que muchas palabras que sólo aparecen en un documento terminaban con un valor de 0 en la celda correspondiente de la matriz para el resto de documentos.

Por tal motivo se implementó el cálculo con TF-IDF que usa normalización normal 0.5. Con este nuevo método todos los términos poseen valores entre 0.5 a 1, eliminando las matrices dispersas.

Y, de la misma forma que hicimos con los otros cambios, se agregó esta funcionalidad a parámetros de entrada, los cuales recomendamos usar de la siguiente manera:

- **Normalización logarítmica**, con documentos que contienen información con poco ruido, esto es, textos de noticias, o provenientes de usuarios que publiquen información coherente

de cualquier índole. Para el cálculo del TF en este método se utilizó la siguiente fórmula:

$$1 + log(f_{t,d}) \quad (4, 3)$$

- **Normalización normal**, cuando exista demasiado ruido en los textos, ej.: de redes sociales; o, cuando se posea demasiados documentos por cada corpus, más de 100. Para el cálculo del TF en este método se utilizó la siguiente fórmula:

$$0.5 + 0.5 \left(\frac{f_{t,d}}{max\{ t\prime \in f_{t\prime,d}\} f_{t\prime,d}} \right) \quad (4, 4)$$

CONCLUSIONES Y RECOMENDACIONES.

Conclusiones.

En el presente trabajo se implementó y probó de manera exitosa un sistema que permita a usuarios sin extensos conocimientos informáticos, procesar y visualizar la evolución de tópicos extraídos de varios tipos de datos.

Enfocados en la portabilidad de la misma, hemos podido reafirmar el potencial de procesamiento que han alcanzado los aplicativos web que se ejecutan en un navegador web; por tal motivo es importante decidir qué sección del procesamiento se ejecuta en el lado del cliente y que sección en el lado del servidor.

En cuanto al procesamiento de información proveniente de redes sociales, es importante la identificación y eliminación del ruido obtenido de la misma, ya que, si no se controla esta problemática, los resultados finales pueden ser nada parecidos a los esperados.

Recomendaciones.

Respecto a los parámetros de entrada del algoritmo JPP, es importante la comparación de resultados obtenidos modificando cada uno de los mismos. En nuestro caso particular podemos afirmar que un valor alpha entre 0 y 1 es ideal para corpus provenientes de redes sociales, los cuales tienen una gran cantidad de ruido.

Para el desarrollo de aplicativos con un alto uso de operaciones matemáticas, se tiene que tomar en cuenta la capacidad de cómputo y memoria disponible que en promedio se requerirán en la ejecución de las mismas. Para esta recomendación es imperativo la ejecución de pruebas de estrés en etapas de desarrollo.

Hacer uso del paradigma “dividir y conquistar” cuando se modela sistemas de procesamiento de datos, esto para evidenciar que secciones pueden ejecutarse en el lado del cliente y cuales en el lado del servidor.

BIBLIOGRAFÍA

[1] Andreas Weiler, Michael Grossniklaus, Marc H. Scholl, “The Stor-e-Motion Visualization for Topic Evolution Tracking in Text Data Streams”, 2015.

[2] C. Vaca, A. Mantrach, A. Jaimes, M. Saerens, “A Time-based Collective Factorization for Topic Discovery and Monitoring in News”, 2014.

[3] E. Carvajal, “Generic Web Worker”, Liberia de JavaScript, site: https://github.com/fercarvo/GenericWebWorker 2018.

[4] Neal Schaffer. Twitter Followers vs Following: What is the Ideal Ratio? [Online]. Disponible en: https://maximizesocialbusiness.com/twitter-followers-following-quality-or-quantity-807