UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE FACULTAD

Departamento de Departamento

Sistema escalable para la detección de necesidades en escenarios de catástrofe natural

Esteban Andrés Abarca Rubio

Profesor guía: Nicolás Hidalgo Castillo

Profesor co-guía: Erika Rosas Olivos

Tesis de grado presentada en conformidad a los requisitos para obtener el grado de Grado

Santiago – Chile

2016

Dedicado a...

AGRADECIMIENTOS

Agradezco a

RESUMEN

Twitter es una red social que cuenta con millones de usuarios en todo el mundo y, en Chile alcanza cerca de los 1.700.000 accesos diariamente. Sus usos van desde ser un microblog personal a la entrega de información o comunicación entre pares. Es por ello que en épocas de necesidad, como lo es el periodo inmediato luego de la ocurrencia de una catástrofe natural, las personas tienden a publicar sus experiencias dentro de éste servicio.
Teniendo en cuentra lo anterior es que se propone un sistema capaz de recoger la información desde *Twitter* — los denominados *tweets* — y procesarla a fin de detectar si es que un *tweet* corresponde a uno en el que el usuario haga mención a uno de los tipos de necesidad que el sistema será capáz de detectar y, finalmente, hacer uso de la información implícita (contenido del *tweet* o metadatos) para presentar la necesidad como un punto en un mapa geográfico del país, de modo que la información obtenida pueda ser tomada por las autoridades correspondientes para que, de esta forma, facilite el proceso de toma de desiciones en cuanto al envío de ayuda a una determinada area dado las necesidades expresadas por la población.
Para lograr lo expuesto anteriormente se ha construido un sistema cuya principal característica está en identificar, a partir de la información contenida en un *tweet* si este hace referencia o no a una necesidad y a dónde corresponde. (**PRINCIPALES RESULTADOS Y CONCLUSIONES ACA**).

Palabras Claves: Palabras; Claves

ABSTRACT

Twitter is a social network that already has millions of users worldwide and, in Chile, reach about of 1.7 millions of accesses daily. Its uses range from being a personal microblog up to information delivery and comunication between peers. It's because of this that in emergencies, such as the inmediate period after a natural catastrophe, people tends to post their experiences on this service.

With this in mind is that is proposed a system able to get information from *Twitter* — as *tweets* — and process it to detect if a user's *tweet* mentions one of the needs that the system can handle and, finally, use the implicit information in the tweet (metadata) and render the need as a geographical position in country's map, thus authority can use the given information and ease the desition making process of sending help to affected areas with the information given by the population.

To achieve these statements previously exposed has been made a system whose main characteristics are identify, by the *tweet*'s given metadata, if it references a need and where it corresponds.

Keywords: Key; words

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE ALGORITMOS

CAPÍTULO 1. INTRODUCCIÓN

1.1 ANTECEDENTES Y MOTIVACIÓN

1.1.1 Motivación

Los desastres naturales en el país han sido frecuentes en los últimos años. Sólo por mencionar algunos de los más recientes y recordados: la erupción del volcán Chaitén (Mayo, 2008), terremoto en Tocopilla (Noviembre, 2007), terremoto Concepción (2010), incendio de las Torres del Paine (Diciembre, 2011), incendio en Valparaíso (Abril, 2014), erupción volcán Villarrica (Marzo, 2015), aluviones en el norte (Marzo, 2015) entre otros. Dependiendo de las características de la emergencia, surgen en la población diversos tipos de necesidades; alimentos, agua, luz eléctrica, refugio, rescate o comunicación. Muchas veces éstas pueden no ser detectadas por las autoridades, al menos, no de forma expedita, lo que no resulta beneficioso para las personas que intentan sobrellevar de la mejor manera posible la crisis y esto se complica aún más cuando la necesidad involucra una necesidad básica, como la falta de agua, donde la vida de los afectados puede correr riesgo. No es problema sólo para las autoridades, Imran, Castillo, Diaz y Vieweg (2014)**CITA CON BIB ACA!** señalan que el comportamiento humano ante crisis como éstas no es de quedarse esperando o huir en pánico, sino que intentan tomar decisiones rápidas en base a la información que conocen. Esto quiere decir que existe gente dispuesta a ayudar, aun siendo ellos los mismos afectados, pero no siempre disponen de la información necesaria para saber dónde apuntar sus esfuerzos. Sería útil, dado lo anterior, tener algún medio que concentre las necesidades que pueda tener una población dentro del país para acudir en su auxilio, posterior a la ocurrencia de una emergencia catastrófica como las mencionadas anteriormente.

1.1.2 Estado del arte

En primer lugar se estudia el actual estado del arte en temas de detección para posteriormente continuar con clasificación y finalmente, sistemas de procesamiento de streams.

Referido a detección, en particular, detección de eventos en medios sociales es señalado por Nguyen & Jung (2015) como el foco de investigación en los últimos años. Sobre la detección propiamente, los autores se refieren a una técnica común consistente en realizar

paquetes de mensajes que son capturados a intervalos regulares de tiempo y luego son procesados, mencionan la existencia de un trade-off entre el intervalo y el performance del sistema al tener que procesar más o menos datos según corresponda. Una segunda alternativa es el procesamiento conforme los datos son recibidos, para ello es necesario una alternativa que permita procesar elementos en paralelo introduciendo problemas de sincronización y balance de carga.

Weng & Lee (2014) señalaron dos principales problemáticas a ser resueltas en Twitter como son el volumen de información generada y la abundancia de información que puede ser catalogada como ruido.

Es por lo anterior que autores como Van De Voort (2014) señala, y es respaldado por la metodología KDD, que, antes de comenzar el proceso de detección, la información recogida ha de ser pre-procesada debido a la mencionada existencia de ruido y el hecho de que los mensajes no siempre cumplen con reglas, ya sean, gramaticales u ortográficas o, también, que, en algunos casos, remover los stopwords o las URL o Hashtag facilita la tarea de análisis a posteriori. Van De Voort también señala las dos formas de detección de eventos existente: Detección en retrospectiva, es decir, capturar los datos y procesarlos posteriormente siguiendo el modelo tradicional de batch processing. Y detección online, es decir, en tiempo real. Tanto la detección como la categorización de eventos pueden realizarse de diversas maneras, por ejemplo el uso de n-gramas, prediciendo la posible siguiente palabra para así llevar a cabo la detección, Rahmound&Elberrichi (2007), otra forma es el uso de técnicas de aprendizaje de máquina (Machine learning, ML) para detectar cuándo un mensaje hace mención a una necesidad y ubicarla dentro de un determinado grupo por medio de un modelo generado a partir de datos de entrenamiento Kamath & James (2011).

Referido a las técnicas de aprendizaje de máquina, Maldonado (2012) presentó un modelo capaz de detectar sentimientos en tweets habiendo utilizado ML aplicando clasificadores Naïve Bayes y Support Vector Machine (SVM), los cuales diferían sólo en la complejidad en realizar la configuración. Wladdimiro & González (2014) haciendo uso de Naïve Bayes y una bolsa de palabras fueron capaces de llevar a cabo una clasificación de necesidades dentro de determinadas categorías. Utilizaron un grupo de personas que se dedicó a clasificar, previamente, un conjunto de datos de entrenamiento para lograr validar el modelo obtenido. Wladdimiro y González propusieron un grafo para modelar el problema de detección de necesidades en tiempo real basado en cinco tareas, pero no consideraron lo mencionado por Van De Voort con respecto a la previa limpieza de datos, presentando problemas de procesamiento innecesario y de-

jaron pendientes problemas relacionados con la carga del sistema y cuellos de botella en su grafo.

El sistema de detección y clasificación desarrollado por estos últimos fue soportado por Apache S4, framework de computación distribuida que, actualmente, se encuentra estancado y sin mantenimiento desde el año 2012. En su lugar Apache Storm se presenta como un framework de computación distribuida para computación en tiempo real distribuida y tolerante a fallas, además, éste promete, una mayor performance y simplicidad de configuración que S4.

1.2 DESCRIPCIÓN DEL PROBLEMA

Se requiere hacer uso de la información generada por la población en *Twitter* para que, en caso de alguna emergencia de carácter nacional, prestar apoyo a las autoridades encargadas de la toma de decisiones, por ejemplo, darles a conocer en qué lugar en particular se requiere asistir a la población con un determinado tipo de ayuda según la necesidad que se presente. ¿Cómo puede usarse la información disponible en Twitter para que, en casos de emergencia, ésta sea útil para ir en directo beneficio de la población en la que se generó satisfaciendo la necesidad específica que presentan?

1.3 SOLUCIÓN PROPUESTA

Se propone una aplicación que estará recogiendo constantemente, en tiempo real, publicaciones desde *Twitter* y analizando si corresponde o no a una necesidad existente en el conjunto de necesidades detectables y, en casos afirmativos, mostrarlas durante un intervalo de tiempo sobre un mapa geográfico del país haciendo uso de los metadatos asociados al tweet, en el caso en que se encuentren disponibles o hacer uso del contenido para inferir sus ubicaciones si es posible.

Al tratarse de una aplicación que recogerá grandes cantidades de información, el desempeño que ésta tendrá ha de ser considerado, por ello, se hará uso de un framework de computación distribuida para procesar las grandes cantidades de tweets de la manera más eficiente posible. Lo anterior quiere decir que la aplicación tendrá, internamente, forma de grafo

dirigido; cada nodo de este grafo corresponderá a un operador por el que la información fluirá. Estos operadores serán aquellos que la literatura señale como los apropiados para el caso, por ejemplo: filtro de *stopwords*, filtro de *spam*, corrector ortográfico, detector de sentimientos, etcétera.

El grupo RESPOND de la Universidad de Santiago de Chile se ha adjudicado fondos para el desarrollo de un proyecto de dos años de duración el cual consiste en el desarrollo de una plataforma de streaming a escala nacional, enfocada en el procesamiento de datos en caso de crisis. Esta plataforma hará uso de la información generada por los usuarios en redes sociales como fuente de datos. Se espera que esta plataforma provea de herramientas para que cualquier persona pueda desarrollar nuevas aplicaciones para atender las diversas problemáticas que puedan existir cuando el país se enfrente a catástrofes.

Para ayudar a difundir la plataforma se requiere construir tres aplicaciones, una que apoye la coordinación de voluntarios, una segunda que difunda noticias y mensajes y, finalmente, una que permita detectar necesidades de la población, todas ellas al presentarse escenarios de catástrofes naturales.

En particular, para este trabajo, se espera atacar el problema de la detección de necesidades de la población y servir de apoyo para la construcción de la plataforma de streaming en relación a qué operadores se han de construir y cómo ha de estructurarse el sistema para operar sobre datos nacionales.

1.4 OBJETIVOS Y ALCANCE DEL PROYECTO

1.4.1 Objetivo general

Construir un sistema escalable para la detección de necesidades de la población en tiempo real para escenarios de desastre natural haciendo uso de *Twitter*.

1.4.2 Objetivos específicos

1. Implementar un método encargado de la recolección de tweets generados dentro del territorio nacional haciendo uso de la API pública de Twitter.
2. Especificar la taxonomía de las necesidades que serán detectadas.
3. Diseñar e implementar el clasificador de necesidades.
4. Definir de los elementos de procesamiento para la construcción del sistema capaz de trabajar los datos obtenidos a gran escala.
5. Implementar una arquitectura escalable que soporte la aplicación.
6. Evaluar la aplicación bajo condiciones de alto tráfico como podría ser el caso de una emergencia nacional.

1.4.3 Alcances

Se utilizarán las publicaciones de *Twitter* para llevar a cabo el procesamiento de la información y no se considera, en el marco de este trabajo, el uso de una red social alternativa, no porque no sea posible, sino que con el motivo de acotar el problema.

Las necesidades que la aplicación detectará no serán todas del universo posible de necesidades existentes, sino de un subconjunto que se considere más importante tanto por el equipo que está trabajando en el proyecto FONDEF como por el profesor patrocinador de éste trabajo; agua, vivienda o luz eléctrica, por ejemplo. De esta forma se logra acotar el problema reduciendo la cantidad de categorías y permitir una mayor precisión en la clasificación (trabajos similares han bordeado una precisión entre el sesenta y ochenta por ciento, pero esto va de la mano con la cantidad de datos utilizados para entrenar), entendiendo la precisión como la relación de elementos clasificados correcta o incorrectamente.

Se considera para la construcción del clasificador un conjunto de cuatro millones setecientos mil *tweets* recogidos desde *Twitter* correspondientes al terremoto ocurrido el 2010 en Chile. Este conjunto de datos contiene mensajes en distintos idiomas y ha sido filtrada llegando a aproximadamente un millón y medio de tweets; de aquel conjunto se obtendrá un subconjunto para realizar el etiquetado y ser usado como datos de entrenamiento, se han etiquetado ya de

2000 a 10000 tweets utilizados para una experiencia anterior. Para dicho etiquetado se utilizará la misma herramienta usado en aquella ocasión: una aplicación donde a voluntarios se les entrega una porción de *tweets* para que sean etiquetados, se les entregan los mismos *tweets* a diferentes personas para no sesgar la muestra a una sola.

La aplicación podrá ser probada en cuando a su performance ante situaciones de gran carga como lo sería el caso de una emergencia, haciendo uso de *JMeter*, herramienta escrita en Java diseñada para realizar tales labores que permitirá simular, entonces, condiciones de alto tráfico dentro de la aplicación por medio del envío de peticiones a la aplicación.

1.5 METODOLOGÍA Y HERRAMIENTAS UTILIZADAS

1.5.1 Metodología

Este trabajo considera dos partes, la primera es la generación del clasificador y la segunda la construcción de la aplicación, donde la segunda depende de haber completado la primera, por ello la principal prioridad será desarrollar este clasificador.

Para realizar el clasificador se tiene considerado el proceso de KDD Fayyad, Piatetsky-Shapiro, y Smyth (1996), acrónimo de Knowledge Discovery in Databases o, simplemente, Descubrimiento (o extracción) de conocimiento en bases de datos. Se refiere al "proceso no-trivial de descubrir conocimiento, patrones e información potencialmente útiles dentro de los datos contenidos en algún repositorio". Este proceso iterativo diseñado para explorar grandes volúmenes de datos. Consta de cinco fases:

- Selección de datos: Se determinan las fuentes de datos y el tipo de información a utilizar. Se extraen los datos útiles de las fuentes de datos.
- Pre-procesamiento: Los datos se preparan y limpian. Se utilizan estrategias para rellenar los datos en blanco o con información faltante. Finalmente en esta etapa se obtiene una estructura de datos adecuada para ser transformada, posteriormente.
- Transformación: Consiste en el tratamiento preliminar de datos, transformación y generación de nuevas variables a partir de las ya existentes.

- Data Mining: Fase de modelamiento propiamente tal. Se utilizan métodos para obtener o detectar patrones que están “ocultos” en los datos.
- Interpretación y evaluación: Se identifican los patrones y se analizan por alguna métrica y se evalúan los resultados obtenidos.

En segundo lugar se tiene la aplicación propiamente tal que será dividida en dos, por un lado se tendrá la aplicación que llamaremos el núcleo que se encargará de recepcionar la información y el visualizador que la mostrará por pantalla.
Para realizar lo anterior se hará el uso de *Extreme Programming* (en adelante XP), presentando avances semanales y discutiendo cambios a realizar en la aplicación, tanto visuales como de funcionamiento interno.
Para manejar las tareas se hará uso de un tablero kanban de cuatro columnas: ”Por hacer”, ”Haciendo”, ”Por Revisar” y ”Completo” donde una tarea sólo podrá considerarse completa habiendo pasado por la revisión y haber sido aceptada.

1.5.2 Herramientas de desarrollo

1.5.2.1 Herramientas de Software

Se han se utilizar las siguientes herramientas de software para la construcción de la aplicación:

- Java como lenguaje de programación principal.
- Apache Storm (1.0.1), como *framework* de computación distribuida.
- Apache Zookeeper (3.4.8), como herramienta para mantener la configuración
- Weka (3.8.0), como herramienta de *Data Mining* para la construcción del clasificador.
- MongoDB (3.2.6), para la persistencia de datos.

- Play Framework (2.5.3), como *framework* para el desarrollo de aplicaciones Java. En particular, la construcción de la aplicación que permitirá visualizar los datos.
- Sublime Text 3 (Build 3103), como editor de textos.
- MiKTex (XeLaTeX), para la escritura de la memoria.
- PowerDesigner 16, para la elaboración de diagramas.
- Bitbucket (Git), como repositorio de todo lo referente al proyecto (Núcleo, Visualizador y Memoria).
- Windows 10 Home Edition (x64).
- Linux Mint 17.3 (x86).
- Oracle VirtualBox (5.0.14).

1.5.2.2 Herramientas de hardware

Se utilizará el equipo del autor cuyas características son las siguientes:

- Procesador Intel Core i5 2.2 Ghz.
- 8 GB de memoria RAM.
- 1 TB de disco duro.

1.6 ORGANIZACIÓN DEL DOCUMENTO

A continuación se presentan a grueso modo los capítulos que componen el presente documento:

El Capítulo Marco Teórico presenta una serie de definiciones detalladamente para ayudar a comprender de mejor manera el problema y los elementos utilizados para su resolución.

El capítulo Construcción aplicaciones detalla los aspectos de toma de requerimientos y diseño de la aplicación detallando las desiciones que se tomaron en ese aspecto.

CAPÍTULO 2. MARCO TEÓRICO

2.1 MINERÍA DE DATOS

A veces llamada como ”descubrimiento de información o conocimiento”, es el proceso de analizar información de diferentes perspectivas y transformarlo en información de utilidad. Puede ser aplicado a distintas fuentes de datos como: bases de datos, imágenes, internet, etc. Es un campo multidiciplinal que involucra el aprendizaje de máquina, la estadística, bases de datos, la inteligencia artificial y la recuperación de información.

Siendo distintos los usos que pueden dársele se pueden generalizar cuatro etapas:

- **Determinación de objetivos**: Delimitar los objetivos que se esperan alcanzar con el proceso de minado.
- **Preprocesamiento de datos**: Se refiere a la limpieza, reducción y transformación de las bases de datos. Es, generalmente, el subproceso que utiliza la mayor cantidad de tiempo.
- **Determinación del modelo**: Aplicación de algoritmos para generar un modelo que cumpla los objetivos planteados. Se genera nuevo conocimiento o se descubre un patrón.
- **Análisis de los resultados**: Se verifica si el conocimiento es útil.

Dentro de las tareas que pueden realizarse utilizando *data mining* pueden encontrarse tales como: Aprendizaje supervisado o clasificación, no supervisado o clustering y reglas de asociación.

2.1.1 Minería de la Web

La aplicación de la minería de datos al contenido que se encuentra en línea es conocida como Minería de la *web* o *web mining*. Se diferencia de la minería de datos tradicional en que ésta última utiliza repositorios de datos; en cambio, la minería *web*, hace uso de información extraída directamente desde la *web*.
Sus métodos son similares en cuanto a sus etapas:

- Selección de las fuentes: referencia al proceso de recuperación de los datos.
- Selección y pre-procesamiento: Incluye cualquier transformación o pre-procesamiento que puedan realizárseles a los datos, por ejemplo, eliminar elementos, como palabras, aplicación de correctores de datos, etc.
- Generalización: Etapa donde se realiza el proceso de minería en sí.
- Análisis: Desarrolla técnicas para utilizar o visualizar el conocimiento adquirido.

La información obtenida puede ser utilizara para analizar tanto el contenido de la web (*Web content mining*) como sus enlaces (o relaciones) (*Web structure mining*) y/o el registro de navegación de los usuarios (*Web usage mining*).

La primera se refiere a búsqueda entre documentos *web* (texto o imágenes), es decir, analiza los documentos y no la relación entre ellos.

La segunda se dedica a analizar la topología de los vínculos existentes y/o analizar la estructura interna de la página *web* y descrbir el *HTML* o el *XML* de la misma.

En particular dentro de la minería de contenido encontramos la minería de texto o *text mining*. Ésta tiene como objetivo el descubrir nueva información a partir colecciones de documentos de texto no estructurado, es decir, texto libre (lenguaje natural, generalmente), aunque también es aceptable otro tipo de información textual como un código fuente. Lo más habitual es trabajar el texto para categorizarlo (Asignar una o más categorías a un documento), clasificarlo (Asignar sólo una clase a un documento) y/o agruparlo (organizar en torno a una jerarquía basado en alguna similitud).

El primer paso para comenzar a trabajar haciendo uso de minería de texto es representar los datos de alguna manera para luego dárselo a los algoritmos adecuados. Algunas de estas representaciones pueden ser las siguientes:

- Bolsas de palabras (*Bag of words*): Representar el texto como un vector de largo n, donde n corresponde al número de palabras, así cada palabra corresponde a un elemento del vector.
- Frases: Considera el texto, simplemente, como una frase sintáctica. Así se permite conservar el contexto.
- N-gramas: Consideran la información de la posición de la palabra en el texto mediante secuencias de longitud n (n-gramas).

Habiendo realizado la representación, el paso siguiente es reducir el conjunto de características. La literatura indica que los métodos más frecuentados son la eliminación de palabras que no aportan información, llevar las palabras a una palabra raíz (*stemming*), entre otros.

2.2 APRENDIZAJE SUPERVISADO

Se caracteriza por ser un proceso de aprendizaje en el que éste se realiza mediante un entrenamiento controlado por un agente externo, el que determina qué respuesta debería generarse a partir de una entrada determinada.

Se asocia al concepto de *machine learning* con la minería de datos; la primera busca patrones conocidos y predecir en base a ellos mientras que la segunda busca patrones con anterioridad desconocidos, es decir, la primera tiene una función focalizada en la predicción mientras que la segunda realiza una función exploratoria.

Los datos son denominados instancias, ejemplares, casos o vectores, donde una instancia corresponde a cada uno de los datos disponibles para el análisis.

Los datos poseen atributos son los elementos dentro de las instancias. Una instancia puede tener asociado un elemento de otro conjunto de atributos llamado ”Clase”, correspondiente a etiquetas de identificación.

Teniendo en cuenta los elementos vistos con anterioridad, se define el objetivo del proceso de aprendizaje como construir una función que relacione las instancias con las clases llamada modelo o, en este caso, clasificador.

Se le llama conjunto de entrenamiento al conjunto de datos utilizado para el aprendizaje. Este conjunto es entregado como entrada al algoritmo de aprendizaje y construcción del modelo. Para realizar la evaluación de la calidad del modelo se utiliza un segundo conjunto de instancias llamado datos de validación. Se espera que estos datos no hayan sido vistos con anterioridad por máquina y así obtener la confianza, es decir, la probabilidad de acierto que calcula el sistema para cada predicción.

Lo ventajoso de este método es que se podrá clasificar una instancia sin haberla visto

nunca, pero la desventaja principal es la que han de utilizarse una gran cantidad de instancias para el proceso de entrenamiento.

El proceso de entrenamiento y evaluación se ilustra en la Figura 2.1.

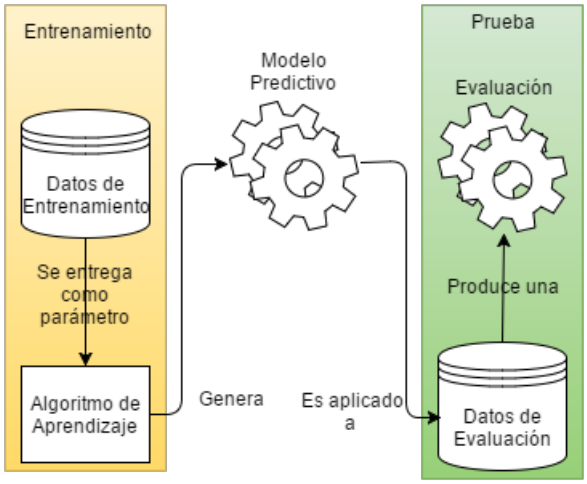

Figura 2.1: Proceso de entrenamiento y prueba del modelo.

Dentro de los algoritmos utilizados para la construcción de clasificadores se encuentran *Naïve Bayes* y SVM (*Support Vector Machine*), a continuación se realiza una descripción de ambos.

2.2.1 Naïve Bayes

La clasificación puede verse como una función γ que asigna etiquetas a observaciones, es decir:

$$\gamma : (x_1, ..., x_n) \rightarrow \{1, 2, ...r_0\}$$

Existe una matriz de costo $cos(r, s)$ con $r, s = 1, ..., r_0$ en el cual se refleja el costo asociado a las clasificaciones incorrectas. En concreto $cos(r, s)$ indica el costo de clasificar un ejemplo de la clase r como de la clase s. En el caso especial de la función de pérdida 0/1, se tiene:

$$cos(r, s) = \begin{cases} 0 & \text{si } r \neq s \\ 1 & \text{si } r = s \end{cases}$$

Subyacente a las observaciones suponemos la existencia de una distribución de

probabilidad conjunta:

$$p(x_1, ..., x_n, c) = p(c|x_1, ..., x_n)p(x_1, ..., x_n) = p(x_1, ..., x_n|c)p(c)$$

La cual es desconocida. El objetivo es construir un clasificador que miniza el coste total de los errores cometidos, y esto se consigue (Duda y Hart, 1973) por medio del clasificador de Bayes:

$$\gamma(x) = argmin_k \sum_{c=1}^{r_0} cos(k, c)p(c|x_1, ..., x_n)$$

En el caso que la función de pérdida sea 0/1, el clasificador de Bayes se convierte en asignar al ejemplo $x = (x_1, ..., x_n)$ la clase con mayor probabilidad a posteriori. Es decir:

$$\gamma(x) = argmax_c p(c|x_1, ..., x_n)$$

En la práctica la función de distribución conjunta $p(x_1, ..., x_n, c)$ es desconocida, y puede ser estimada a partir de una muestra aleatorea simple $\{(x^{(1)}, c^{(1)}), ..., (x^{(N)}, C^{(N)})\}$ extraida de dicha función de distribución conjunta.

2.2.2 SVM

2.3 METODOLOGÍA

2.3.1 Programación Extrema

La Programación Extrema (*Extreme Programming*, XP desde ahora en adelante), comenzó como un proyecto el 6 de Marzo de 1996. Es uno de los procesos ágiles más populares y ha sido provado exitosamente en compañias e industrias de todos los tamaños **?**.

Su éxito se debe a que hace especial incapié en la satisfacción del cliente por sobre la entrega de todo lo el software posible.

Aporta cinco formas escenciales para mejorar el proceso de desarrollo de software: Comunicación, simplicidad, retroalimentación, respeto y coraje: Constantemente se comunica al equipo de desarrollo con el cliente. Se intenta mantener el diseño lo más simple y sencillo posible. Se obtiene retroalimentación desde las pruebas desde el día uno. Se les entrega el software al cliente lo más pronto posible con los cambios solicitados. El éxito depende, en gran medida, del respeto y comunicación de los miembros del equipo y los clientes. Implementando XP el equipo puede responder a los cambios sin temor.

La metodología implementa unas simples reglas de trabajo, las que se dividen en cinco grandes áreas las que se detallarán a continuación.

1. Planeación:
    - Se escriben Historias de usuario.
    - Se crea un plan de *releases*.
    - Se planifican liberaciones pequeñas y frecuentes.
    - Se divide el proyecto en iteraciones.
    - Al comienzo de cada iteración se planea cómo será.
2. Manejo:
    - Se le da al equipo una área de trabajo.
    - Se realizan reunione del tipo *stand up meeting* a díario.
    - Se mide la velocidad del proyecto.
    - Se mueven a las personas de sus puestos (para que todo el equipo pueda trabajar en todo).
    - Se solucionan problemas que instroduzcan quiebres en la metodología.
3. Diseño:
    - Simplicidad. El mejor diseño es el más simple.
    - Se crean *spikes* para reducir el riesgo.
    - No se agregan funcionalidades antes de tiempo.
    - Hacer uso de técnicas de *refactoring*, cada vez que sea posible.
4. Implementación:
    - El cliente siempre está disponible.

- El código debe ser escrito bajo estándares.
- Se hace uso de *Test Driven Development* (TDD).
- Todo el código debe hacerse haciendo uso de *pair programming*.
- Sólo una pareja integra código a la vez.
- Integración a menudo.
- Se cuenta con un equipo dedicado a la integración.
- El código es de todos.

5. Prueba:
    - Todo el código debe tener pruebas unitarias.
    - Todas las pruebas deben ser pasadas antes de una liberación.
    - Cuando se encuentra un *bug*, se crean pruebas.
    - Los *test* de aceptación se corren a menudo y sus resultados son publicados.

Éstas reglas por si solas pueden carecer de sentido, pero se apoyan en los **valores** que la metodología quiere entregar y que fueron mencionadas anteriormente, pero ahora son detalladas:

- Simplicidad: Se hará lo que se solicitó, pero no más. Ésto maximiza el valor entregado dado una fecha límite. Nuestras metas se alcanzarán por medio de pequeños pasos para mitigar errores tan pronto ocurran. Crearemos algo de lo que estemos orgullosos y lo mantendremos en el tiempo a costos razonables.
- Comunicación: Todos somos partes de un equipo y nos comunicamos cara a cara a diario. Trabajaremos juntos en todo: desde la toma de requerimientos hasta la implementación. Crearemos la mejor solución posible al problema.
- Retroalimentación: Cada iteración será completada seriamente entregando *software* funcional. Mostraremos nuestro *software* a menudo y prontamente para luego escuchar y aplicar los cambios solicitados. Hablaremos de nuestro proyecto y adaptaremos nuestro proceso a el, no al revéz.
- Respeto: Todos dan y reciben el respeto que merecen como miembros del equipo. Todos contribuyen con valor así sea simple entusiasmo. Los desarrolladores respetan la experiencia del cliente y viceversa.
- Coraje: Diremos la verdad sobre el progreso y nuestras estimaciones. No se documentan excusas por si se falla porque se planea tener éxito. No tenemos porque no trabajamos solos. Nos adaptaremos a los cambio cuando ocurran.

El proceso de XP puede puede ser aprecido en la Figura 2.2.

Figura 2.2: Diagrama de flujo de Programación Extrema

2.3.2 *Knowledge Discovery in Databases* (KDD)

Es definido por Fayyad (1996) como ”El proceso no trivial de identificar patrones válidos, nuevos, potencialmente útiles y en ultima instancia comprensible en los datos”, surge de la necesidad de manejar grandes cantidades de datos e involucra simultaneamente varias disciplinas de investigación tales como el aprendizaje automático, la estadística, inteligencia artificial, sistemas de gestión de bases de datos, sistemas de apoyo a la toma de desiciones, entre otras.

Si bien puede variar el usuario, quien es aquel que determina el domino de la aplicación, es decir, cómo se utilizarán los datos, el proceso generalmente considera las siguientes etapas:

1. Selección de datos: Consiste en buscar el objetivo y las herramientas del proceso de minería, identificando los datos que han de ser extraídos, buscando atributos apropiados de entrada y la información de salida para representar la tarea. Esto quiere decir, primero se debe tener en cuenta lo que se sabe, lo que se quiere obtener y cuáles son los datos que nos facilitarán esa información para poder llegar a nuestra meta, antes de comenzar el proceso como tal.
2. Limpieza de datos: En este paso se limpian los atributos sucios, incluyendo datos incompletos, el ruido y datos inconsistentes. Estos datos sucios, en algunos casos, deben ser eliminados, pues pueden contribuir a un análisis inexacto y resultados incorrectos.

3. Integración de datos: Combina datos de múltiples procedencias incluyendo múltiples bases de datos, que podrían tener diferentes contenidos y formatos.

4. Transformación de datos: Consiste en modificaciones sintácticas llevadas a cabo sobre los datos sin que suponga un cambio en la técnica de minería aplicada. Tiene dos caras, por un lado existen ventajas en el sentido de mejorar la interpretación de las reglas descubiertas y reduce el tiempo de ejecución, por el otro puede llevar a la pérdida de información.

5. Reducción de datos: Reducción del tamaño de los datos, encontrando características más significativas dependiendo del objetivo del proceso.

6. Minería de datos: Consiste en la búsqueda de patrones de interés que puedan expresarse como un modelo o dependencia de los datos. Se ha de de especificar un criterio de preferencia para seleccionar un modelo de un conjunto de posibles modelos. Además se ha de especificar la estrategia de búsqueda (algoritmo), a utilizar.

7. Evaluación de los patrones: Se identifican patrones interesantes que representan conocimiento utilizando diferentes técnicas incluyendo análisis estadísticos y lenguajes de consulta.

8. Interpretación de resultados: Consiste en entender resultados de análisis y sus implicaciones y puede llevar a regresar a algunos pasos anteriores.

La representacíón del proceso puede verse en la Figura 2.3.

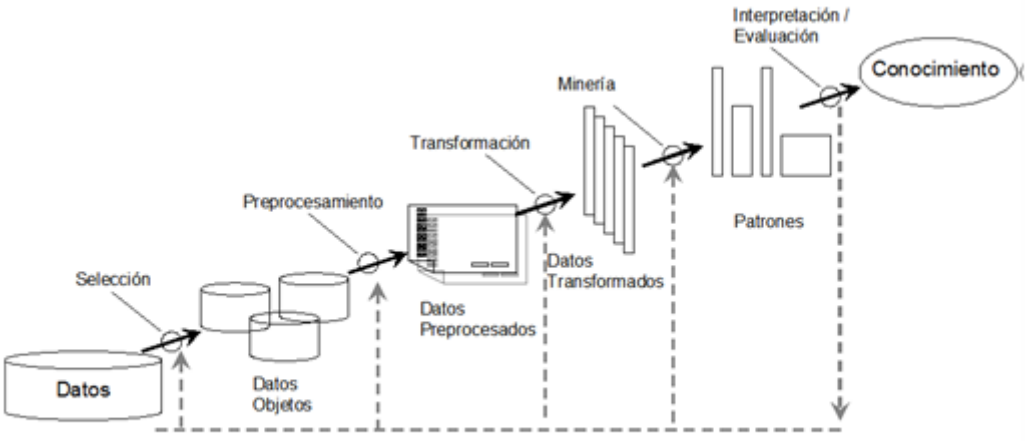

Figura 2.3: Proceso KDD.

2.4 HERRAMIENTAS

2.4.1 Play Framework

Es un *framework* de código abierto para aplicaciones *web* escrito en *Java* y *Scala*, el cual sigue el patrón de arquitectura *Modelo-Vista-Controlador* (MVC). Utiliza el paradigma de diseño ”Convención sobre configuración”, el cual apunta a reducir la toma de desiciones que debe tomar el desarrollador sin perder flexibilidad.

Se enfoca en la productividad a aplicaciones *RESTful*.

Elimina la desventaja de desarrollo al utilizar *Java* dada por el continuo ciclo de compilar-empaquetamiento-despliegue. Al detectar cambios en el código realiza inmediatamente la compilación y actualiza en la JVM sin necesidad de reiniciar el servidor.

Play no utilizar sesiones en su funcionar, privilegiando el uso de almacenamiento *offline* o el uso de peticiones *Ajax* para resolver problemas del lado del cliente.

2.4.2 Apache Storm

Apache Storm es un sistema de computación en tiempo real de código abierto. Simplifica el problema de flujos (*streams*), de datos sin que estos tengan fin.

Es escalable, tolerante a fallos y garantiza que toda la información será procesada. Presenta *Benchmarks* que señalan que por nodo es capaz de procesar más de un millón de tuplas por segundo.

Se compone principalmente de dos partes. La primera es denominada *Spout* y es la encargada de recoger el flujo de datos de entrada. La segunda es denominada *Bolt* y es la encargada de la transformación o procesado de los datos.

Oficialmente es representado como puede verse en la Figura 2.4. donde los *Spouts*

son representados simulando ser llaves de agua desde donde fluyen los datos al sistema y los *Bolts* como rayos donde se procesa el flujo.

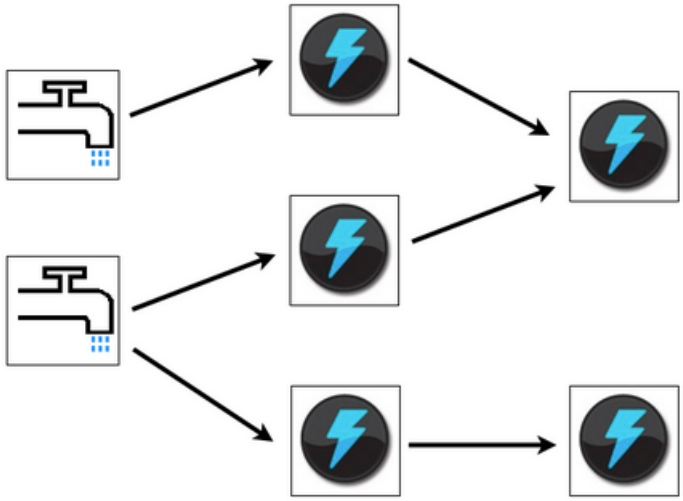

Figura 2.4: Representación del funcionamiento de Apache Storm.

Uno de los puntos fuertes que tiene este sistema es que al crear una topología donde se instancian *Bolts* y *Spouts*, Storm se encarga de escalar el sistema distribuyendo los elementos en sus componentes.

2.4.2.1 Topología

Una topología de Storm es similar a un grafo. Cada nodo se encarga de procesar una determinada información y le pasa el testigo al siguiente nodo. Está compuesta por *Spouts* y *Bolts*.

2.4.2.2 Cluster de Storm

Un cluster de Storm no muere, se queda siempre en espera de nuevos datos de entrada mientras el proceso siga activo.

La arquitectura de Storm se divide en tres componentes:

- Master Node: Ejecuta el demonio llamado Nimbus, el cual es responsable de distribuir el código a través del cluster. Realiza la asignación y monitorización de tareas en las distintas máquinas del cluster.
- Worker Node: Ejecutan el demonio Supervisor, el cual se encarga de recoger y procesar los trabajos asignados en la máquina donde está corriendo. En caso de fallo de uno *Worker Node*, Nimbus se dará cuenta y redirigirá el trabajo a otro.
- Zookeeper: Si bien no es un componente propio de Storm, es necesario para su funcionamiento, pues se encarga de coordinar Nimbus y Supervisor, además de mantener sus estados, pues ambos son *stateless*.

2.4.2.3 Modos de funcionamiento

Storm puede funcionar de dos modos: Local y Cluster. El primero es útil para el desarrollo, pues ejecuta toda la topología en una única JVM, por lo que pueden realizarse fácilmente pruebas de integración, depurar código, etcétera. Este modo simula, haciendo uso de *Threads*, cada nodo del Cluster.

El modo Cluster es considerado el 'modo de producción' y es el modo donde el código es distribuido en máquinas diferentes dentro del Cluster.

2.4.2.4 Stream grouping

Se refiere a la forma en la que se van a compartir los datos entre los componentes. Como modelo de datos, Storm utiliza tuplas que son listas de valores con un nombre específico. El valor puede ser cualquier tipo, para ello se ha de implementar un serializador.

- Shuffle grouping: Storm decide de forma aleatoria la tarea a la que se va a enviar la tupla, de manera que la distribución sea equivalente entre todos los nodos.

- Fields gruoping: Se agrupan los *streams* por un determinado campo de manera que se distribuyen los valores que cumplen una determinada condición a la misma tarea.
- All grouping: El *stream* pasa por todas las tareas haciendo multicast.
- Grobal grouping: El *stream* se envía al *bolt* con ID más bajo.
- None grouping: Es un *Shuffle grouping* donde el orden no es importante.
- Direct grouping: La tarea es la encargada de decidir hacia donde emitir especificando el ID del destinatario.
- Local grouping: Se utiliza el mismo *bolt* si tiene una o más tareas en el mismo proceso.

2.4.3 MongoDB

Base de datos no relacional (NoSQL) de código abierto escrita en C++ y está orientada al trabajo en documentos. Lo anterior quiere decir que, en lugar de guardar los datos en registros, lo hace en documentos y éstos son almacenada en una representación binaria de JSON conocida como BSON.

Una de las diferencias fundamentales con respecto a las bases de datos relacionales es que no es necesario que se siga un esquema; en una misma colección - concepto similar a una tabla en las bases de datos relacionales - pueden tener distintos esquemas.

MongoDB fue creado para brindar escalabilidad, rendimiento y disponibilidad. Puede ser utilizado en un servidor único como en múltiples. Esto se logra dado que MongoDB brinda un elevado rendimiento, tanto para lectura como para escritura, potenciando la computación en memoria.

Las consultas en MongoDB se realizan como si se tratase de Javascript entregando como parámetro un objeto JSON. Por ejemplo, dado el documento presentado en la Figura 2.5, parte de una colección llamada ’Personas’ en MongoDB:

Una consulta para encontrar este elemento dentro de la colección se daría de la forma aprecida en la Figura 2.6

En pruebas realizando operaciones habituales dentro de las bases de datos el **?** demostró que el tiempo de ejecución de MongoDB, como base de datos NoSQL, aventaja

Figura 2.5: Documento en MongoDB.

```
db.Personas.find({Nombre:"Juan"});
```

Figura 2.6: Consulta en MongoDB.

significativamente a las bases de datos relacionales más populares como lo son MySQL y PostgreSQL.

CAPÍTULO 3. CONSTRUCCIÓN APLICACIONES

Este capítulo detalla la fase de construcción de la aplicación, comenzando con la toma de requistos, la toma de desiciones de diseño y culmina con la implementación.

3.1 TOMA DE REQUISITOS

Se han construido dos aplicaciones; La primera es el visualizador, el que se encarga de aplicar los filtros y mostrar la información correspondiente a los puntos donde se han detectado necesidades y la segunda corresponde a aquella que realiza la recepción del flujo de datos desde *Twitter* y su paso por el clasificador. Las razones por las que se consideraron dos aplicaciones separadas se especificarán en la sección 3.2. A continuación se presenta el proceso de diseño y construcción de cada una de ellas.

3.1.1 Visualizador

En conversaciones con los profesores guía y co-guía del presente trabajo - Clientes - se señaló que se requería un visualizador en el que se mostraran las necesidades recogidas desde *Twitter*. Para ello se sugirió utilizar un mapa en el cual se señalara mediante marcadores los puntos en cuestión, de modo que éste fue el punto inicial.

La Tabla 3.1. lista las historias de usuario que representan los requisitos identificados en el presente. Estas hisotorias tienen un ID y una descripción, donde el ID se compone de ”HU-vXX”, haciendo referencia a que corresponde al visualizador, donde ’XX’ corresponde al número del requisito.

Estos requisitos no fueron producto de sólo una reunión con los clientes, sino que son producto de una serie de demostraciones de la aplicación, sugeridos como cambios deseables a la aplicación y fueron considerados de acuerdo a la metodología.

Tabla 3.1: Tabla de historias de usuario.

ID	Descripción
HU-v01	Como usuario quiero visualizar las necesidades expresadas como un punto en un mapa geográfico del país.
HU-v02	Como usuario quiero poder filtrar qué necesidades mostrar en cada momento según su categoría.
HU-v03	Como usuario quiero que, según el nivel de acercamiento del mapa, los puntos se agrupen.
HU-v04	Como usuario quiero que el agrupamiento pueda realizarse según su categoría en lugar de su proximidad.
HU-v05	Como usuario quiero seleccionar un intervalo de tiempo y que se muestren los puntos identificados dentro de ese intervalo.
HU-v06	Como usuario quiero poder especificar términos para la búsqueda de necesidades.
HU-v07	Como usuario quiero que los parámetros para el funcionamiento del sistema sean modificables.
HU-v08	Como usuario quiero que se muestre la nomenclatura de los íconos que son mostrados en el mapa.
HU-v08	Como usuario quiero que se visualicen estadísticas sobre la cantidad de elementos procesados como por ejemplo: cantidad de tweets, necesidades detectadas y cantidad de usuarios diferentes.

3.1.2 Clasificador

Como parte de la aplicación la función del clasificador se pensó desde el primer momento el ser soportado por Apache Storm, aquel fue el punto de inicio; La aplicación internamente tendría forma de grafo y cada nodo de aquel grafó — *Bolt* — realizará una tarea específica.

Dentro de la metodología KDD, como se señaló en la sección 2.3.2 un paso de preprocesamiento de los datos, por lo que se consideró que la información proveniente desde *Twitter* debería sufrir el mismo preprocesamiento que señala **??**. Según lo anteriormente descrito se originan los requisitos que dan como resultado las historias de usuario que se presentan en la Tabla 3.2. Junto con la descripción la ID de la historía tendrá la siguiente forma: ”HU-cXX”, donde ’XX’ corresponde al número del requisito.

Tabla 3.2: Requisitos del clasificador.

ID	Descripción
HU-c01	Como cliente quiero que la aplicación esté soportada sobre Apache Storm.
HU-c02	Como cliente quiero que la aplicación recoja el flujo de datos de Twitter.
HU-c03	Como cliente quiero que se apliquen los filtros descritos en la literatura.
HU-c04	Como cliente quiero que se consiga la ubicación desde dónde se originó el Tweet o del lugar al que hace referencia.
HU-c05	Como cliente quiero conocer la necesidad a la que el Tweet hace referencia, si es que lo hace.

3.2 DESICIONES DE DISEÑO

Como se mencionó en la sección anterior, se haría uso de Apache Storm como *framework* de computación distribuida para asegurar la escalabilidad del sistema. Dado el uso de Apache Storm (Con su versión 0.10 disponible al momento de la realización de este trabajo), la elaboración de una aplicación única que realizase ambas labores: procesar y visualizar, se dificultaba, dado que la aplicación que utilizase Storm como base requería ser ejecutada utilizando la aplicación distribuída por Apache además de Zookeeper, mientras que para implementar una aplicación web requeriría utilizar un servidor de aplicaciones. Es por ello que se decidió elaborar dos aplicaciones que, en conjunto, suplieran las necesidades del cliente.

Teniendo claro que consistiría de dos aplicaciones separadas surgía el problema de cómo realizar la comunicación entre ellas.

Por el lado del visualizador *web* restaba saber qué *framework* sería el más apropiado para su desarrollo. Dado que el equipo de desarrollo está limitado a una persona se pensó en un *framework* que permitiera agilizar el proceso de desarrollo; por ello en lugar de otro se consideró el uso de *Play Framework*, dado su enfoque en equipos de desarrollo ágil y su rapidez para visualizar los cambios del código **??**.

En un primer momento se pensó en realizar la comunicación REST dado que las aplicaciones desarrolladas mediante *Play Framework* son, por defecto, RESTful, mas no se consideró implementar persistencia en los datos para que pudiesen ser considerados con posterioridad. Luego de discutir el diseño con los clientes se llego a la conclusión que la implementación de un sistema de base de datos permitiría elaborar la historia de usuario HU-v05.

Finalmente restaba seleccionar qué motor de base de datos. Se pensó en MySQL en primer lugar dado su familiaridad y simplicidad, pero se desechó al considerar que se trabajará con grandes cantidados de datos lo que involucrará muchas operaciones IO de escritura y lectura; dato el trabajo presentado en **??** además de la información de rendimiento presentada en **??**, se seleccionó MongoDB como base de datos para el sistema.

Teniendo ya todos los elementos para construir la aplicación se presentó la arquitectura presente en la Figura 3.1. En ella se aprecia que através de la API de *Twitter* la aplicación "Clasificador" (siendo ejecutara dobre Storm), recibe el *stream* para posteriormente procesarlo y almacenarlo dentro de la base de datos. Por su parte el "Visualizador" recoge la información desde la base de datos y la muestra por pantalla. Además es posible pasarle términos de búsqueda al clasificador haciendo uso de la misma base de datos.

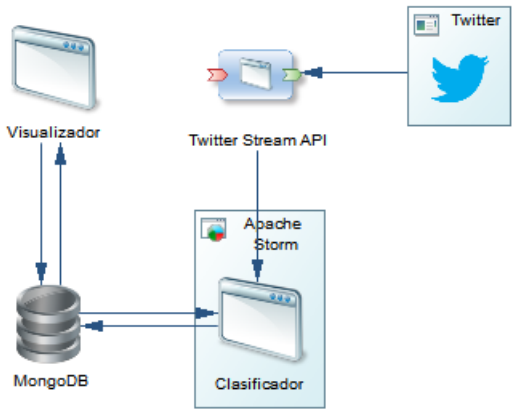

Figura 3.1: Arquitectura general del sistema.

Si bien no es necesario, se definieron dos esquemas para almacenar la información en la base de datos: El primero corresponde al esquema que tendrá la información guardada por el clasificador correspondiente a los marcadores y la segunda corresponde a las consultas realizadas desde el visualizador como términos para filtrar la búsqueda dentro del *stream*. Dichos esquemas pueden ser apreciados en los ejemplos presentados en las Figuras 3.2 y 3.3, respectivamente.

Figura 3.2: Ejemplo de documento en la colección Markers.

Donde ”id” corresponde a la ID generada por MongoDB al almacenar el documento, ”contenido” corresponde al texto original del *Tweet*, ”categoría” corresponde, como su nombre lo señala, a la categoría a la que pertenece (agua, alimento, electricidad, comunicación, personas, seguridad o irrelevante), ”latitud” y ”longitud” corresponden a las coordenadas geográficas decimales donde se ubica el marcador, ”userID” corresponde al ID del usuario que emitió ese *Tweet* y, finalmente, ”generatedAt” al *timestamp* de cuándo fue generado ese marcador.

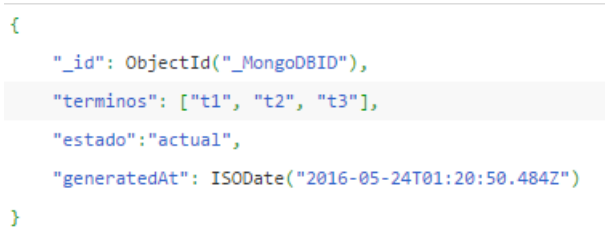

Figura 3.3: Ejemplo de documento en la colección Markers.

De igual manera que en el documento de marcador, ”id” corresponde a la ID generada por MongoDB al almacenar, ”terminos” corresponde a un arreglo de *string* que almacena los términos de búsqueda para filtrar el *stream* de entrada, ”estado” puede tomar dos valores: ”actual” o ”antigua” y sirve para señalar si se debe o no filtrar por los términos del documento. Finalmente ”generatedAt” corresponde al *timestamp* de cuándo fué originada esa consulta.

CAPÍTULO 4. EXPERIMENTACIÓN

4.1 BLABLA

CAPÍTULO 5. RESULTADOS

5.1 BLABLA2

CAPÍTULO 6. CONCLUSIONES

Lorem ipsum dolor sit cuchuflí barquillo bacán jote gamba listeilor po cahuín, luca melón con vino pichanga coscacho ni ahí peinar la muñeca chuchada al chancho achoclonar. Chorrocientos pituto ubicatex huevo duro bolsero cachureo el hoyo del queque en cana huevón el año del loly hacerla corta impeque de miedo quilterry la raja longi ñecla. Hilo curado rayuela carrete quina guagua lorea piola ni ahí (Samza, 2014).

REFERENCIAS BIBLIOGRÁFICAS

Codish, M., Marriott, K., & Taboch, C. (2000). Improving program analyses by structure untupling. *Journal of Logic Programming*, *43*, 251–263.

Samza, A. (2014). Samza. [Online] `http://samza.incubator.apache.org/`.

ANEXO A. ANEXO DE EJEMPLO

Lorem ipsum dolor sit cuchuflí barquillo bacán jote gamba listeilor po cahuín, luca melón con vino pichanga coscacho ni ahí peinar la muñeca chuchada al chancho achoclonar. Chorrocientos pituto ubicatex huevo duro bolsero cachureo el hoyo del queque en cana huevón el año del loly hacerla corta impeque de miedo quilterry la raja longi ñecla. Hilo curado rayuela carrete quina guagua lorea piola ni ahí.