

Araujo García José Mauricio

Osorio Hernández Noé

Prof. Ivan Olmos Pineda

Minería de Datos

Primer Examen Parcial

Introducción 3

Problemática 3

Análisis 4

Base de Datos 4

Infraestructura 6

Aplicación 7

Desarrollo 8

Conclusiones 13

Referencias 14

# Introducción

Antes de que se puedan utilizar los algoritmos de minería de datos, se debe ensamblar un conjunto de datos objetivo. Como la minería de datos solo puede descubrir patrones presentes en los datos, este conjunto debe ser lo suficientemente grande como para contener estos patrones y ser lo suficientemente conciso como para extraerse dentro de un límite de tiempo aceptable. El preprocesamiento es esencial para analizar los conjuntos de datos multivariados antes de la minería de datos. Es por esto que es importante realizar una recolección adecuada que sirva para los propósitos que se quieren y no dificulte sobremanera o limite el análisis que se llevará acabo.

En el presente documento se realizará el análisis de un problema particular de la vida real y se expondrá una solución desde la etapa de recolección de datos hasta el análisis de un modelo de clasificación.

# Problemática

El bosque es considerado como uno de los recursos más importantes e indispensables y los incendios forestales representan una amenaza constante para los sistemas ecológicos, la infraestructura y los aspectos ambientales de una comunidad, la detección de incendios forestales es un tema muy importante en la supresión previa proceso. Esto da lugar a la necesidad urgente de detectar incendios forestales lo más rápido y eficiente posible.

# Análisis

*Data Sets*

Las imágenes se encuentran divididas en tres categorías: 'incendio', 'no incendio' y 'humo' con un total de alrededor de 6000 imágenes. Estas imágenes son principalmente de bosques o entornos similares. Las imágenes etiquetadas como 'incendio' contienen llamas visibles, las imágenes de 'humo' contienen humo que indica el inicio de un incendio. Finalmente, las imágenes etiquetadas como 'no incendio' son imágenes de bosques.

*Preprocesamiento de Imágenes*

*Métricas de Imágenes*

Se creó un script para extraer información acerca de los segmentos de las imágenes en diferentes tipos de estadísticas. Para escribir el script, se ocuparon las librerías de python*: scipy, PIL, numpy, pandas y opencv.* Estas librerías facilitan el manejo de imágenes y la obtención de características de la imagen que se utilizaron para calcular las siguientes métricas: blancura, uniformidad, color dominante, probabilidad de color dominante por canal, color promedio, dimensiones, borrosidad, entropía por canal, desviación estándar por canal, coeficiente de variación por canal.

Aplicación

La estructura de la parte de aplicación es sencillo. Se siguió el paradigma REST, o Representational State Transfer para comunicar al cliente y al servidor. Un sistema REST se caracteriza por su falta de estado, ya que separa completamente al cliente y al servidor desde la implementación. Esta arquitectura separa completamente el código del cliente y del servidor, por lo que ni siquiera tienen que estar escritos en el mismo lenguaje. Además, cada uno de estos componentes se hace modular, escalaban y mantenible. Si el código del cliente cambia, no afecta al servidor y vice-versa, siempre y cuando se mantenga la estructura de la comunicación entre ellos.

En esta arquitectura, el cliente manda solicitudes para crear, modificar, leer y borrar información, y el servidor le contesta con respuestas a las solicitudes hechas. La manera de comunicarse es mediante verbos HTTP (comúnmente GET, POST, PUT, DELETE) que llaman a rutas específicas del servidor (aplicación/usuarios/1029).

# Desarrollo

Lo siguiente es exponer la solución dada. Se decidió desarrollar una aplicación web ya que este tipo de aplicaciones le son familiares a la mayoría de personas de casi todas las edades. El primer paso del desarrollo fue desarrollar una interfaz para la aplicación donde se pudieran crear y aplicar los instrumentos, así como administrar a los usuarios. Esta interfaz fue hecha con la librería Vue.js*,* los resultados se presentan a continuación.

Para soportar y servir los llamados de la aplicación, se implementó un servidor web en el lenguaje Go con el router HTTP gorilla/mux y el driver de mongo-go-driver. Este servidor se encarga de recibir y responder a todas las solicitudes del cliente, y de interactuar con la base de datos.

Teniendo la aplicación funcional, se empaquetó la aplicación en un contenedor de Docker mediante un DockerFile, el cual compila la aplicación y el servidor, instala todo lo necesario para que funcionen e inicia el scheduler supervisord, que se encarga de manejar estos dos procesos dentro de la aplicación. El contenedor tiene la siguiente estructura.

La aplicación y el servidor se pueden ver como cualquiera de las aplicaciones App A-

F en la Fig. 2.5. Se creó una imagen de este contenedor que fue almacenado y publicado en Docker Hub, para facilidad de uso dentro del clúster.

Finalmente, teniendo todos los componentes necesarios, se levantó el clúster de Kubernetes con un único plano de control en tres computadoras diferentes, todas con el sistema operativo de CentOS. Para levantar un clúster, lo primero es crearlo utilizando la herramienta *kubeadm* e iniciando el servicio *kubelet* en cada uno de los nodos. Una vez creado, es necesario instalar una red definida por software (SDN, Software Defined Network) para poder comunicar a los nodos dentro del clúster. El SDN elegido fue Project Calico por su licencia abierta y escalabilidad.

Se probó la comunicación de los nodos, y cuando ésta fue exitosa, se procedió a desplegar el contenedor mediante un Replica Set de Kubernetes. Se creó el despliegue mediante un archivo de configuración YAML, en el que se especificó el número de replicas deseadas (en este caso 3, una por cada nodo). Al crear el despliegue, se levantan múltiples pods (igual al número de replicas especificadas) y Kubernetes se encarga de repartirlos entre los nodos.

Después de confirmar que los pods estén funcionando dentro de cada nodo, se expuso el despliegue mediante un servicio, el cual regresa la dirección IP del clúster para poder acceder a la aplicación. La aplicación dentro del clúster tiene una estructura como lo muestra el siguiente diagrama.

# Conclusiones

La solución propuesta cumple con los requisitos del problema, y lo hace de una manera eficaz y rápida. Pero esta solución es una prueba de concepto solamente, no es apropiada para un despliegue en producción, ya que no se consideraron muchas configuraciones extras que no son completamente necesarias en un ambiente de desarrollo. Sin embargo, se prueba que una aplicación distribuida transparente, tolerante a fallos, escalable y además amigable para los usuarios es enteramente posible y no está fuera de las posibilidades de ningún estudiante ni desarrollador. Esta tecnología fue desarrollada por las empresas líderes en la industria para solucionar los problemas que el cómputo en la nube presenta y que ahora son herramientas de código abierto accesibles para cualquiera. Es por esto que en el futuro, muy probablemente se verán aplicaciones que utilicen una arquitectura distribuida de este tipo para hacerle frente al creciente número de usuarios.

# Referencias

1. Araujo, M. (2019, Noviembre 26). mauricioaraujo / psytest. Recuperado Diciembre 2, 2019, de <https://hub.docker.com/repository/docker/mauricioaraujo/psytest/general>.
2. Creating a single control-plane cluster with kubeadm. (2019, June 12). Recuperado Diciembre 2, 2019, de <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/>.
3. Docker Documentation. (n.d.). Recuperado Diciembre 2, 2019, de <https://docs.docker.com/>.
4. Pedersen, B. E. (2018, Mayo 5). Replica Set. Recuperado Diciembre 2, 2019, de <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>.
5. Perry, S. (2018, Mayo 15). Using Minikube to Create a Cluster. Recuperado Diciembre 2, 2019, de <https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/>.
6. Quickstart for Calico on Kubernetes. (n.d.). Recuperado Diciembre 2, 2019, de <https://docs.projectcalico.org/v3.10/introduction/>.
7. Replication. (2008). Recuperado Diciembre 2, 2019, de <https://docs.mongodb.com/manual/replication/>.
8. Silverlock, M. (2016, Julio 18). gorilla/mux. Recuperado Diciembre 2, 2019, de <https://github.com/gorilla/mux>.
9. Supervisor: A Process Control System. (2004). Recuperado Diciembre 2, 2019, de <http://supervisord.org/>.
10. Trejo, A. (2013, Agosto 25). Clasificación de los instrumentos psicológicos. Recuperado Diciembre 2, 2019, de <https://www.cognicionpsicologica.com/clasificacion-de-los-instrumentos-psicologicos/>.
11. What is REST. (2017). Recuperado Diciembre 2, 2019, de <https://restfulapi.net/>.
12. Wilson, C. (2017, Mayo 26). MongoDB Go Driver. Recuperado Diciembre 2, 2019, de https://github.com/mongodb/mongo-go-driver.