ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE INFORMÁTICA

MÁSTER EN DATA SCIENCE

TRABAJO FIN DE MÁSTER

VISUALIZACIÓN STREAMING DE MODELO PREDICTIVO DE DATOS DE RED

Autor: Ignacio Arias Barra

Tutor: José Felipe Ortega Soto

Curso académico: 2016/2017

RESUMEN

El trabajo presentado a continuación recoge el diseño, implementación y pruebas de un sistema de visualización en *streaming* [20]. El principal objeto de visualización consistirá en la evaluación de un modelo de predicción de tráfico cursado a través de la red.

Con vistas en el desarrollo en el mundo del Big Data [17], poco a poco van siendo más demandados sistemas de análisis de grandes cantidades de datos que tomen decisiones o calculen resultados en tiempo real.

Estos modelos tienen como objetivo la optimización de recursos, así como la posibilidad de encontrar nuevas formas de explotación de los propios.

En el presente trabajo se ha realizado la unión de varias de estas nuevas tecnologías y métodos de modelaje y análisis de datos.

AGRADECIMIENTOS

El presente trabajo de fin de máster ha sido realizado bajo la supervisión del profesor José Felipe Ortega Soto al que le quisiera agradecer el interés mostrado y todo el tiempo empleado en motivarme y hacer que todo lo sufrido durante el desarrollo y las pruebas hayan merecido la pena.

A Leticia Martín Martín, por darme todo su apoyo en los días en los que la jornada laboral se unía al desarrollo del trabajo.

A mi hermana Lucía Arias Barra, por mostrar su afecto e interés por el trabajo que he realizado.

A mis padres José Ignacio Arias Castro y María Jesús Barra Bermejo, por hacerme la vida más fácil durante el proyecto y mostrar su interés en que lo finalizara.

A mis amigos, que me han ayudado a evadirme y a disfrutar en los momentos difíciles.

ÍNDICE GENERAL

CAPÍTULO I

Introducción

La principal motivación de este trabajo es el desarrollo de un sistema de visualización de datos en *streaming* [20], en el cual se apliquen técnicas de *data science* [18] que se encuentran a la orden del día.

En el mundo tecnológico actual, la generación de datos por cualquier dispositivo se encuentra en un crecimiento constante, prácticamente exponencial. Esto genera la necesidad de crear nuevas técnicas de procesado y modelado de datos en grandes cantidades, así como nuevas tecnologías para llevarlo a cabo.

El procesado de estos datos se puede llevar a cabo de dos maneras: una con conjuntos estáticos de información y otra con conjuntos de datos dinámicos. La primera forma reúne un conjunto de técnicas en el que se hace más hincapié en el modelo que se quiere obtener. En este caso, podría ser necesitarse tener máquinas potentes con la capacidad de cómputo suficiente para el análisis de grandes cantidades de datos y posiblemente sea necesaria una arquitectura distribuida o clúster de máquinas. Sin embargo, en la segunda forma, se ha de prestar atención además a otro tipo de factores que aumentan la complejidad de la estructura de procesado, ya que dicha estructura tiene que mantener un flujo constante de obtención, transformación, envío y modelado de datos. En este caso el almacenamiento pasa a un segundo plano. Este último entorno será el que se desarrollará en el presente trabajo.

El principal interés en este trabajo reside en juntar la aplicación de técnicas de procesado, modelado y visualización de datos en *streaming* [20]. Como principales puntos se destacan:

- Creación de entorno multi-tecnológico.
- Utilización de modelos de Machine-Learning (ML) [19] en *streaming* [20].
- Modelo de predicción de datos de navegación por la red.

El funcionamiento básico consiste en un usuario, que genera tráfico por internet proveniente de diferentes fuentes de información (diario deportivo, descarga de vídeo y simulación de llamadas sobre VoIP). El tráfico que generan ha sido analizado con la finalidad de crear un modelo de predicción. Cuando el usuario cursa un tráfico determinado, el modelo tratará de predecir de qué fuente provienen los datos y con la visualización podremos ver las estadísticas de acierto del modelo entre otros datos de interés.

1.1 Estructura de la memoria

- Capítulo I "Introducción": este capítulo se muestra la principal motivación por la que se ha propuesto el sistema, así como los puntos que se han de tener en cuenta en arquitectura de procesado de datos en *streaming* [20].
- Capítulo II "Descripción del *dataset*": este apartado se describe cómo se ha obtenido el conjunto de datos así como las variables del dataset.
- Capítulo III "Arquitectura general": este capítulo detalla el funcionamiento, así como la estructura, las tecnologías empleadas y los parámetros del sistema.
- Capítulo IV "Caso de estudio": en este apartado se explican los experimentos realizados. Además, se muestran los resultados obtenidos durante la puesta en marcha de todos ellos y las dificultades encontradas.
- Capítulo V "Conclusiones y líneas futuras": aquí se exponen los conocimientos adquiridos durante el desarrollo e implementación del sistema y los conocimientos de los que se partía antes de comenzar. Una vez puesto en marcha, se realiza una evaluación de los resultados obtenidos con los que se extraen las conclusiones finales sobre el funcionamiento del sistema. Para finalizar, se realiza un resumen a modo de visión futura, de lo que podrían ser algunas mejoras del sistema para dar cabida a un mayor rango de casuísticas que tener en cuenta.

CAPÍTULO II

Descripción del dataset

2.1 Descripción general

Una parte importante del presente trabajo consta de la clasificación del tráfico cursado por un usuario. Para ello, ha sido necesario una fase previa de entrenamiento del modelo. En cada tipo de tráfico, se ha generado un *dataset* debidamente etiquetado, mediante un periodo de generación y recolección de datos durante 10 minutos. Después se han juntado los 3 conjuntos en uno con una extensión de ~150k registros (aproximadamente 50k registros por tipo de tráfico) y localizado en el directorio *src/Data/data_traffic* del repositorio (APÉNDICE A).

2.2 Fuentes de datos

El trabajo consta de un carácter demostrativo, es decir, la finalidad del mismo no es sólo estudiar si es posible realizarlo sino también hacer una demostración práctica de ello. Debido a esto, la parte de la generación de datos se ha automatizado con los siguientes scripts:

src/TrafficGenerator/trafficGenerator.py: script Python [4] encargado de llamar a cada tipo de generador de tráfico en función de los parámetros de su configuración.

src/TrafficGenerator/ generator.sh: script de shell bash [5] que contiene los diferentes tipos de generadores de tráfico. También se encarga de realizar las llamadas al sistema de los elementos descargados para mantener siempre el sistema limpio.

Mientras se generan los datos, se ha utilizado un analizador de tráfico que fuera capaz de realizar un filtrado de los mismos. Se ha utilizado el TCPDUMP [16], sobre el sistema operativo Ubuntu [3] en el que se ha realizado todo el trabajo. La gestión se ha automatizado con los siguientes scripts:

src/TrafficAnalyzer/StreamingLogAnalyzer.py: script Python [4] que se encarga de gestionar diferentes tipos de analizadores, uno para cada tipo de tráfico. También tiene como misión la gestión de los mensajes recibidos y enviados a Kafka [7].

src/TrafficAnalyzer/tcpdump.py: script Python [4] que directamente ejecuta los analizadores.

A continuación, se detallan las 3 fuentes a partir de las cuales se ha generado el tráfico de red.

Tráfico web browsing

El primer tipo de tráfico consiste en navegación por la página del as. En concreto, en la descarga de la página *www.as.com.*

Tráfico vídeo

El segundo tipo de tráfico consiste en la descarga de un video de la plataforma de *youtube*. En concreto el siguiente, *https://www.youtube.com/watch?v=ruabyha_mGI.*

Tráfico VoIP

Para el tercer tipo de tráfico, se han simulado llamadas sobre VoIP. Para ello, se ha hecho uso de la llamada al sistema *ping,* capaz de generar paquetes a una dirección IP dada. El tamaño de paquete simulado de VoIP tiene las siguientes características:

- Encoder de 64000 bits/second
- VoIP Payload en segundos, por paquete, de 0.02
- Transformación de bit a byte, multiplicando por 8.
- Cabecera IP de 40 bytes
- Cabecera Ethernet de 18 bytes.

La dirección IP a la que se envían los paquetes es la 8.8.8.8. Es el servidor DNS de Google.

2.3 Variables

En este apartado se explicará cada variable y el tipo.

Is_youtube:

Tipo: numérica

Valores: 0,1

Explicación: variable que indica si el paquete procede de tráfico de youtube. Servirá para la evaluación del modelo. 0, el paquete no procede de youtube, 1 sí.

VoIP

Tipo: numérica

Valores: 0,1

Explicación: variable que indica si el paquete procede de tráfico de voIP. Servirá para la evaluación del modelo. 0, el paquete no procede de voIP, 1 sí.

browsing

Tipo: numérica

Valores: 0,1

Explicación: variable que indica si el paquete procede de tráfico de browsing (as.com). Servirá para la evaluación del modelo. 0, el paquete no procede de browsing, 1 sí.

proto

Tipo: categórica

Valores: tcp, ICMP, UDP

Explicación: variable que indica el protocolo al que pertenece el paquete.

IP_SrcIP

Tipo: categórica

Valores: IP origen en formato xxx.xxx.xxx.xxx

Explicación: variable que contiene la IP origen del paquete

IP_DstIP

Tipo: categórica

Valores: IP destino en formato xxx.xxx.xxx.xxx

Explicación: variable que contiene la IP destino del paquete

IP_TotLen

Tipo: numérica

Valores: números enteros

Explicación: variable que contiene los bytes del paquete

IP_Version

Tipo: cateórica

Valores: 4

Explicación: versión del protocolo de IP

IP_Uplink

Tipo: numérica

Valores: 0,1

Explicación: variable que indica el si paquete procede del enlace de uplink (1) o del downlink (0)

TimeStamp

Tipo: timeStamp

Valores: hora

Explicación: variable que contiene la hora a la que se origina el paquete

dpiPktNum

Tipo: categórica

Valores: formato "IP_Uplink-Número de paquete"

Explicación: variable que contiene el flag de subida o bajada del paquete además del número de paquete generado en el sistema.

coord._1

Tipo: numérica

Valores: entre 0 y 180

Explicación: variable que contiene la coordenada longitud del paquete (valor aleatorio)

coord._2

Tipo: numérica

Valores: entre -90 y 90

Explicación: variable que contiene la coordenada latitud del paquete (valor aleatorio)

IP_FiveTuple

Tipo: categórica

Valores: formato "proto + IP_SrcIP + IP_DstIP + SrcPort + DstPort"

Explicación: variable que contiene los valores antes descritos, con la finalidad de poder categorizar los paquetes por esta quintupleta.

CAPÍTULO III

Arquitectura general

3.1 Plataforma

3.1.1 Dispositivo

El dispositivo sobre el que se ha desarrollado la totalidad del presente trabajo ha sido un ordenador portátil con las siguientes características:

Procesador:	Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.59 GHz
Memoria instalada (RAM):	12,0 GB (11,9 GB utilizable)
Tipo de sistema:	Sistema operativo de 64 bits, procesador x64

3.1.2 Sistema operativo (O.S)

El sistema operativo sobre el que se ha desarrollado la totalidad del presente trabajo ha sido un sistema UNIX, Ubuntu en su versión 16.04 [3].

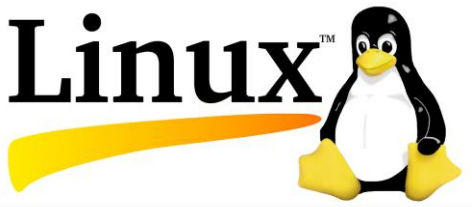

Figura 1. Logo SO Linux.

3.2 Tecnologías

3.2.1 Anaconda

Se define como un entorno de programación para distribuciones de Python, así como un gestor de paquetes. Permite instalar paquetes en entornos separados y

facilitar así un control de versiones de librerías sin que interfieran entre posibles diferentes versiones instaladas en la misma máquina.

Interacción en el sistema

Para el desarrollo del trabajo se ha utilizado un entorno virtual Anaconda [14] con una versión de Python [4] 2.7. Se han instalado todos los paquetes necesarios para la demo, como por ejemplo pyspark, pykafka…

Figura 2. Logo Anaconda [14].

3.2.2 Python

Lenguaje de programación en el que se ha desarrollado la mayor parte del trabajo. Es de fácil aprendizaje y permite trabajar e integrar sistemas de forma rápida y efectiva [4].

Interacción en el sistema

La simplicidad y alto nivel del lenguaje ha permitido crear con mayor facilidad la compleja estructura sobre la que se basa la demo.

Figura 3. Logo Python [4].

3.2.3 Shell bash

Es un programa encargado de interpretar órdenes y realizar acciones en el sistema operativo.

Interacción en el sistema

El lenguaje que se utiliza para la escritura de las órdenes se llama Shell scripting. Ha sido elegido para desarrollar el programa *src/TrafficGenerator/generator.sh*. Este código requiere de interacciones sencillas con el sistema operativo de un modo eficiente, por ello se ha decidido emplearlo en este caso.

Figura 4. Imagen Shell Bash [5].

3.2.4 Docker

Consiste en una plataforma de desarrollo software. El elemento principal de trabajo de Docker son los contenedores. Un contenedor consiste en un programa junto con sus librerías y dependencias del sistema operativo, todo empaquetado y listo para ser ejecutado. Los contenedores pueden ser ejecutados en cualquier máquina que tenga Docker instalado, sin necesidad de tener las librerías o dependencias instaladas en el sistema operativo principal, ya que se encuentran todas en el contenedor Docker que se haya creado.

Existe un repositorio general, el Docker Hub, donde hay multitud de contenedores creados para optimizar los recursos a la hora de tener instalados programas y sus dependencias (ej: Skype, Kafka…)

Interacción en el sistema

Para esta demo se ha utilizado un contenedor Docker desarrollado por los creadores del programa Spotify. El servicio principal que brinda este contenedor es la herramienta Kafka, además de dependencias como la herramienta de organización y gestión llamada Zookeeper.

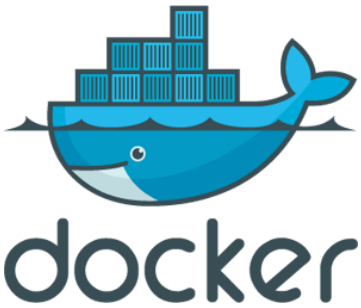

Figura 5. Logo Docker [6].

3.2.5 Kafka

Herramienta que se utiliza para la construcción de estructuras de comunicación de datos entre aplicaciones. Los canales o pipelines que usa son muy útiles para conseguir s*treaming* [20] de mensajes en el sistema.

Interacción en el sistema

El objetivo principal es proporcionar un servicio de envío y recepción de mensajes entre los diferentes módulos de la demo. En todos los módulos del sistema existen transformaciones de datos que necesitan ser comunicadas entre las diferentes partes de forma inmediata, para poder obtener un sistema en tiempo real.

Figura 6. Logo Kafka [7].

3.2.6 Elasticsearch

Elasticsearch [8] es la base sobre la que montar diferentes tipos de servicios y con la que conseguir sistemas en "casi tiempo real". Consiste en una base de datos en la cual la unidad principal de almacenamiento es el documento de formato tipo *json*. Una de las principales ventajas de esta base de datos es que es capaz de inferir esquemas de indexado a partir de los documentos insertados en la misma.

Interacción en el sistema

En la demo, esta base de datos es utilizada como apoyo a la visualización de resultados en *streaming* [20]. Mediante un módulo cliente creado con la librería Elasticsearch [8] de Python [4], se van insertando documentos en la base de datos

con una disponibilidad casi inmediata para la utilización de los mismos por otros servicios.

Figura 7. Logo Elasticsearch [8].

3.2.7 Kibana

Permite la visualización y exploración de datos que se encuentran en Elasticsearch [8]. La fácil integración con la base de datos permite crear las visualizaciones necesarias para la ejecución de la demo.

Interacción en el sistema

Esta herramienta es la que se utiliza para mostrar los resultados de las predicciones del modelo en tiempo real, así como otras métricas.

Figura 8. Logo Kibana [9].

3.2.8 Spark y pyspark

Spark [10] es una herramienta que permite el procesado de grandes cantidades de datos. La principal característica que la hace tan conocida y usada es la utilización optimizada de los recursos del sistema. Se pueden configurar parámetros de utilización de los cores de la máquina, así como a nivel de memoria o disco duro.

El lenguaje principal sobre el que se encuentra escrito es Scala. Sin embargo, tiene intérpretes que hacen que los lenguajes Python [4], Scala y Java puedan desarrollar scripts con instrucciones de Spark. En el presente trabajo se usa Python [4] como se

ha mencionado con anterioridad, por lo que la librería utilizada para Spark, es llamada pyspark.

Interacción en el sistema

Lo que se utiliza de esta herramienta es el motor de entendimiento con la máquina, ya que lo que se usa para el procesado de datos es el módulo de *streaming* [20].

Figura 9. Logo Spark [10].

3.2.9 *Spark Streaming*

Módulo de Spark [10] utilizado para el procesado de los datos. Como se ha comentado anteriormente, el presente trabajo tiene como objetivo la visualización en *streaming* de la evaluación de un modelo. Debido a ello, todas las partes de la demo tienen que tener un carácter de procesado y de envío/recepción de mensajes en tiempo real o semi real.

Interacción en el sistema

Este módulo es usado en el principal punto de transformación de datos. Éste se encuentra en la recepción de los datos generados por el analizador de tráfico TCPDUMP [16]. Recibe toda la información generada por el tráfico de red y la transforma en un formato más fácil de entender para el resto del sistema.

Figura 10. Logo Spark [21].

3.2.10 Spark ML y Spark MLlib

Spark ML [11] y Spark MLlib [12] son dos librerías contenidas en Spark [21]. Ambas tienen como objetivo el facilitar el uso de modelos de machine learning. La tendencia actual de Spark [21] está siendo la de migrar los modelos desde Spark MLlib [11] a Spark ML [12]. En el presente trabajo se han usado métodos de ambas librerías.

Interacción en el sistema

Las dos librerías han sido utilizadas para la creación del modelo de predicción. Para la creación y evaluación del mismo se han utilizado dos formas diferentes. En la primera, llevada a cabo con Spark ML [12], se ha creado y evaluado el modelo utilizando los datos en *batch* del modelo Random Forest [13]. En la segunda forma utilizando Spark MLlib [11], a la hora de realizar predicciones se realizan valor a valor en Random Forest [13] y para los modelos Gradient Boosted Trees [23] y Multilayer Perceptron [24]. Aunque ambas librería presenten muy buenos resultados, esta última será la empleada en el presente trabajo, ya que se necesitará realizar una clasificación paquete a paquete.

Figura 10. Logo Spark MLlib [11].

3.2.11 Jupyter notebook

Consiste en una aplicación web que permite crear documentos que contengan código, texto plano, ecuaciones… Es bastante adecuado para realizar exploraciones sencillas de datos, de tal manera que queden explicadas a la vez que se puede ejecutar código en los notebooks creados.

Interacción en el sistema

Se ha empleado para la exploración de los datos recogidos para la creación del modelo. Como lenguaje de código se ha utilizado Python [4]. Una vez instalado *jupyter* [15] en la máquina, se ha puesto en marcha un servidor en local con el que se ha creado el notebook necesario para el análisis.

Figura 11. Logo Jupyter [15].

3.2.12 TCPDUMP

Consiste en un analizador de tráfico web. Para cada paquete que se produce cuando se está navegando por internet puede llegar a proporcionar gran cantidad de información, como la ip destino, la ip origen, puertos, tamaño de paquete..

Interacción en el sistema

En el presente trabajo se necesita para capturar la información de los paquetes que se generen cuando haya una conexión a cualquiera de las 3 fuentes disponibles.

La ejecución de la herramienta se lleva a cabo con el siguiente comando:

sudo tcpdump -l -s 0 -q -U -n -i

A continuación se detalla la explicación de cada uno de ellos:

sudo*:* comando Linux que nos permite realizar acciones con permisos de administrador

tcpdump*:* nombre con el que se llama a la herramienta

-l: opción que permite usar buffer en la salida de la información. Útil cuando se quiere ver la información mientras se captura.

-**s**: opción en la que se indica el mínimo de bytes de información capturados por paquete. En este caso, con la opción 0, se usa el valor por defecto 262144 bytes.

-**q:** opción que permite mostrar menos información del protocolo por pantalla

-**U:** opción que permite (sin especificar la opción -w) escribir la información capturada en el momento de la captura en lugar de esperar a que el buffer se llene.

-**n:** opción que permite no convertir los puertos y direcciones en nombres y dejar los valores originales (números).

-**i:** opción que permite indicar la interfaz por la que se capture el tráfico producido.

3.3 Esquema y funcionamiento

La siguiente figura muestra la estructura general que se ha desarrollado para poder llevar a cabo la demostración:

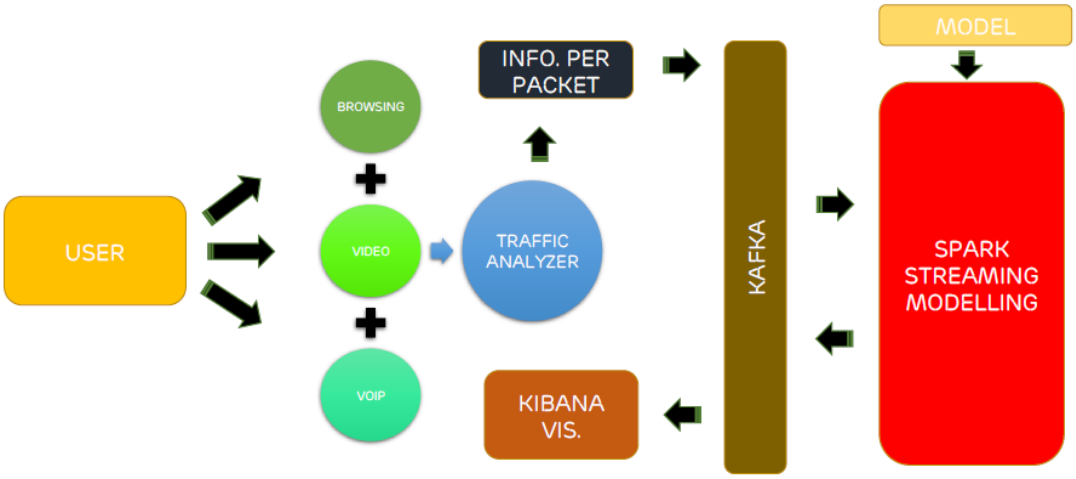

Figura 12. Esquema general.

El funcionamiento es el siguiente:

1º El usuario está generando tráfico de cualquiera de los tres tipos que se indican.

2º Los paquetes generados, son analizados por TCPDUMP [16] en la parte de "Traffic analyzer" a través de Kafka [7].

3º La información generada para cada paquete es analizada por el módulo de Spark Streaming [21] y transformada con la finalidad de extraer la información relevante en el formato deseado.

4º El módulo de Spark Streaming [21] se encarga de usar el modelo creado con anterioridad (versión *streaming* [20]). Realiza la clasificación y a su vez la evaluación de dicha clasificación para cada paquete. Añade dicha información y genera un fichero en formato *json* por paquete, que envía a Kafka [7] de nuevo.

5º Estos mensajes son recibidos por un cliente ElasticSearch [8], elaborado con Python, que recoge los mensajes de Kafka [7] y los introduce en la base de datos.

6º Según se van introduciendo los datos en ElasticSearch [8], las gráficas generadas previamente en Kibana [9], se van actualizando pudiendo observar la evolución del proceso en tiempo real.

3.4 Ejecución de la demo

1º Poner en ejecución el docker Kafka [7]

Para ello, en una terminal ejecutamos el siguiente comando:

"sudo docker run -itd --name=kafka -p 2181:2181 -p 9092:9092 --env ADVERTISED_HOST=localhost --env ADVERTISED_PORT=9092 --env CONSUMER_THREADS=10 spotify/kafka"

En caso de querer parar su ejecución, ejecutaremos:

"docker stop kafka; docker rm Kafka"

2º Poner en ejecución ElasticSearch y Kibana

En la carpeta donde nos hemos bajado los archivos de ElasticSearch ejecutamos:

"./bin/elasticsearch"

Para comprobar que está en funcionamiento, en el navegador ponemos la siguiente dirección: http://localhost:9200. Si obtenemos respuesta, todo estará en orden.

Como se ha comentado con anterioridad, Kibana se ha configurado para que comience con ElasticSearch, por lo que, si escribimos la siguiente dirección en el navegador, http://localhost:5600, podremos ver la interfaz web.

3º Ejecutar cliente Kibana

En la carpeta src/Visualization, ejecutamos

"python kibana_vis.py"

4º Generación de tráfico

La primera vez que se pone en funcionamiento el sistema en el dispositivo, es necesario generar algo de tráfico para que se creen los topics de Kafka [7]. Sin esto, el módulo de Spark Streaming [21] no sería capaz de ejecutarse.

En la carpeta src/TrafficGenerator ejecutar:

"python trafficGenerator.py"

5º Ejecutamos el analizador de tráfico (módulo Spark Streaming [21])

En la carpeta src/TrafficAnalyzer ejecutar:

"python StreamingLogAnalyzer.py"

Se ha de recalcar que este módulo activa por debajo la herramienta TCPDUMP [16], a través del módulo *tcpdump.py*.

6º Generación de tráfico para su clasificación.

Ejecutamos de nuevo el paso 4. Se ha de recalcar que este módulo ejecuta por debajo el script que activa los generadores de tráficos, el *generator.sh*.

7º visualizar las gráficas (dashboard o individuales)

Todos los módulos que hacen uso de Kafka [7], usarán la clase que se ha creado, llamada KafkaConnection.

CAPÍTULO IV

Caso de estudio

4.1 Análisis planteado

Una vez realizada la recolección de los datos, se da paso al análisis de los mismos. La exploración de los datos se lleva a cabo a través de un notebook de Python (Jupyter [15]), bajo un entorno de programación llamado Anaconda [14]. El archivo que contiene el análisis detallado se llama: *src/Analysis/TFM data analysis.ipynb* .

Previamente, se ha llevado a cabo la lectura de algunos papers relacionados con la clasificación de tráfico con técnicas de ML [1] [2]. En ambos hacen referencia a la clasificación del tráfico usando el concepto de flow de paquetes, es decir, el conjunto ordenado de paquetes con un mismo identificador y tomando como referencia el valor del tamaño del paquete. De esta forma, se pueden crear modelos de clasificación con la ayuda de la comparación entre flows o series temporales.

A continuación, se puede echar un vistazo a ciertas características del dataset. La primera de ellas es el número de paquetes generados durante el mismo periodo de recolección de datos.

	proto	IP_TotLen
label		
browsing	38112	38112
is_youtube	56235	56235
voIP	52183	52183

Tabla 1. Número de paquetes recogidos en 10 minutos.

Como se puede observar, la navegación web normal ha generado menos paquetes respecto a las otras dos fuentes de información.

Teniendo en cuenta el tamaño de paquete generado para cada tipo de tráfico (como se indicaba en los papers leídos [1] [2]) se ha generado la siguiente tabla.

	IP_TotLen							
	count	mean	std	min	25%	50%	75%	max
label								
browsing	38112.0	2429.369674	3497.168700	0.0	0.0	133.0	4344.0	27512.0
is_youtube	56235.0	1757.337619	1989.861707	0.0	0.0	1448.0	2896.0	17376.0
voIP	52183.0	175.132629	27.367425	0.0	175.0	175.0	175.0	4155.0

Tabla 2. Descripción de datos, tamaño de paquete.

Si nos fijamos en la columna de la media de tamaño de paquete, columna "mean", se puede observar que son bastante diferentes entre sí, lo que a priori nos da una pista de que la clasificación usando el tamaño de paquete puede estar bien encaminada.

Yendo más allá, se ha decidido usar algún dato adicional para la clasificación que no dependiera del lugar de recolección de datos o de ejecución de la demo (si usásemos la IP origen o destino, la IP desde la que se ejecute la demo va a ir cambiando en función del dispositivo en el que lleve a cabo, por lo que estaríamos introduciendo datos nuevos que no se han tenido en cuenta en el modelo y llevaría a una tasa mayor de error).

La variable elegida ha sido el protocolo al que pertenece cada paquete. Los tres tipos son TCP, UDP e ICMP.

	label	IP_TotLen
proto		
ICMP	51956	51956
UDP	1508	1508
tcp	93066	93066

Tabla 3. Número de paquetes por protocolo.

Se observa que el tráfico minoritario es UDP, mientras que el mayoritario es tcp.

La tabla que vemos a continuación da información acerca del tamaño de paquete en función del protocolo.

	IP_TotLen							
	count	mean	std	min	25%	50%	75%	max
proto								
ICMP	51956.0	175.000000	0.000000	175.0	175.0	175.0	175.0	175.0
UDP	1508.0	83.465517	85.238964	24.0	28.0	35.0	133.0	451.0
tcp	93066.0	2055.882900	2730.843771	0.0	0.0	1448.0	2896.0	27512.0

Tabla 4. Tamaño de paquete por protocolo.

De nuevo el tamaño medio de paquete aparenta un buen método de clasificación del tráfico. En este caso observamos que el tamaño de paquete para ICMP es en todos los percentiles de 175. Esto es debido a que en la simulación del tráfico VoIP, la mayoría de los paquetes son ICMPs generados con el mismo tamaño.

En siguiente vemos el número de paquetes clasificados por tráfico y protocolo.

		IP_TotLen
label	proto	
browsing	UDP	1293
	tcp	36819
is_youtube	UDP	69
	tcp	56166
voIP	ICMP	51956
	UDP	146
	tcp	81

Tabla 5. Número de paquetes por tráfico y protocolo.

Como se ha comentado, el tráfico mayoritario viene en el protocolo tcp, mientras que los paquetes simulados de VoIP, debido a la herramienta ping, son ICMPs. Esto podría suponer otra división a la hora de clasificar el tráfico.

Por último, vemos un barrido por las características de los paquetes clasificados por tráfico y protocolo.

		IP_TotLen							
		count	mean	std	min	25%	50%	75%	max
label	proto								
browsing	UDP	1293.0	82.337974	87.083690	24.0	28.0	56.0	133.0	451.0
	tcp	36819.0	2511.792118	3529.756939	0.0	0.0	226.0	4344.0	27512.0
is_youtube	UDP	69.0	124.275362	84.941136	33.0	42.0	83.0	213.0	246.0
	tcp	56166.0	1759.343838	1990.257499	0.0	0.0	1448.0	2896.0	17376.0
voIP	ICMP	51956.0	175.000000	0.000000	175.0	175.0	175.0	175.0	175.0
	UDP	146.0	74.164384	60.549214	31.0	35.0	35.0	154.0	300.0
	tcp	81.0	442.197531	625.349131	0.0	0.0	257.0	1033.0	4155.0

Tabla 6. Descripción de datos por tráfico y protocolo.

Fijándonos en la media de paquete, podemos observar que los valores tienen claras diferencias entre ello. Esto es buena señal a la hora de clasificar el tráfico.

4.2 Conclusiones sobre el análisis

Una vez vistas las tablas del apartado anterior, se ha optado por utilizar el tamaño de paquete, además del protocolo, como métodos de clasificación, en lugar de los flows sobre los que se hablan en los papers [1] [2].

Tras esta decisión, el siguiente paso ha sido la creación de un dataset a partir del cual generar un modelo. Se ha partido del conjunto de datos y se han eliminado columnas, poniendo la etiqueta del tráfico (columna "label") en primer lugar. Después, se ha utilizado un formato especial para transformar el csv obtenido al formato que necesita Spark [10] en su librería ML [12] (acerca de machine learning) y la librería de Python [4], pyspark, llamado libsvm.

4.3 Realización de pruebas

Este trabajo buscaba desarrollar un sistema de visualización en *streaming* [20] de la evaluación de un modelo. Para ello, se han realizado pruebas de diferentes modelos de clasificación. El primero sobre el que se realizaron pruebas fue el Gradient Boosted Trees [23]. Sin embargo, al ejecutar el modelo, apareció el siguiente error:

```
"Labels must be in {0,1}; note that GBTClassifier currently on
ly supports binary classification."
```

En este caso, tenemos 3 posibles clases sobre las que clasificar, por lo que este método no ha sido válido para la realización de las pruebas.

Los dos siguientes modelos, Multilayer Perceptron [24] y Random Forest [13] si han sido válidos para la realización de pruebas.

4.3.1 Gradient Boosted Trees

Como se ha mencionado con anterioridad, no ha sido válido para la realización de pruebas:

```
"Labels must be in {0,1}; note that GBTClassifier currently on
ly supports binary classification."
```

4.3.2 Multilayer perceptron

El perceptrón multicapa nos da la posibilidad de añadir o quitar el número de capas y neuronas en cada capa que se desee. Por ello se han realizado dos pruebas diferentes.

La primera configuración de capas ha sido de 2-5-4-3 neuronas. Los resultados han sido los siguientes:

- Precisión acierto conjunto de test: 0.525141
- Precisión acierto conjunto entrenamiento: 0.526175

La segunda configuración de capas ha sido de 2-5-4-4-7-3 neuronas. Los resultados han sido los siguientes:

- Precisión acierto conjunto de test: 0.517721
- Precisión acierto conjunto entrenamiento: 0.519133

Como se puede observar, los resultados obtenidos en la precisión de dicho modelo no son muy buenos.

El modelo se puede encontrar en el directorio *src/Model/Multi_percep*.

4.3.3 Random Forest

Por último, se han realizado pruebas con Random Forest [13]. En primer lugar se probó el algoritmo de clasificación del módulo de ML [12] de la librería pyspark, con una evaluación del modelo en batch. Se han realizado pruebas dividiendo el dataset en uno de test y otro de entrenamiento, y a continuación se muestran los resultados.

```
TRAINING DATA
+--------------+-----+-------------+
|predictedLabel|label|     features|
+--------------+-----+-------------+
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
+--------------+-----+-------------+
only showing top 5 rows

Test Error = 0.211873
RandomForestClassificationModel (uid=rfc_053eced0c34b) with 100 trees
```

Tabla 7. Resultado clasificación con datos de entrenamiento.

```
TEST DATA
+--------------+-----+-------------+
|predictedLabel|label|     features|
+--------------+-----+-------------+
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
+--------------+-----+-------------+
only showing top 5 rows

Test Error = 0.215483
RandomForestClassificationModel (uid=rfc_053eced0c34b) with 100 trees
```

Tabla 8. Resultado clasificación con datos de test.

De las tablas anteriores, podemos extraer las siguientes conclusiones:

- Modelo de Random Forest creado con 100 árboles.
- Tasas de error de clasificación muy bajas, por lo que tendremos una buena precisión en el modelo.
- Tasas de error de clasificación bastante parecidas, lo que da lugar a un buen modelo de clasificación.

Este modelo ha sido guardado en el directorio *src/Model/ RandomForest_Batch* del repositorio. Los resultados obtenidos con este modelo son bastante buenos. Se ha de recordar que la forma de creación y validación del modelo ha sido de forma estática. Para este trabajo necesitamos realizar la clasificación de forma dinámica y paquete a paquete.

Debido a esto se ha usado el módulo Mllib [11] de pyspark, que nos permite realizar la clasificación en la forma necesaria. Los resultados de la evaluación del modelo han sido los siguientes:

```
Test Error = 0.214785373608903
Learned classification forest model:
TreeEnsembleModel classifier with 100 trees
```

Se ha obtenido una tasa de error bastante baja y similar a la del modelo en batch. Este modelo ha sido guardado en el directorio *src/Model/ RandomForest_Streaming* del repositorio.

CAPÍTULO V

Conclusiones y líneas futuras

5.1 Resultados

5.1.1 Benchmarking de modelos

En la siguiente tabla se muestra un resumen de los resultados obtenidos en la evaluación de los modelos tanto en test como en training. Lo que se muestra es la precisión en el acierto, a la hora de clasificar los datos.

Modelo	Precisión en training	Precisión en test	Posición por test
Gradient Boosted Trees	0	0	5º
Multilayer Perceptron (4 layers)	0.526175	0.525141	3º
Multilayer Perceptron (6 layers)	0.517721	0.519133	4º
Random Forest ML	0.784517	0.788127	1º
Random Forest MLlib	-	0.785215	2º

Tabla 9. Resultados precisión en la clasificación de los modelos.

Recordemos que el primer modelo sobre el que se realizaron pruebas resultó que no era válido para este problema de clasificación por tener más de 2 clases.

Respecto al resto, se ha obtenido que el modelo Random Forest [13] en batch, es el modelo que ha dado unos mejores resultados. En el lado opuesto tenemos el perceptrón multicapa con 6 capas.

Cabe puntualizar que la precisión para el training con la librería Random Forest [13] de Mllib no se ha realizado por no estar el código obtenido de Spark preparado para ello.

5.1.2 Gráficas de la ejecución

En este apartado se pueden observar gráficas correspondientes a la ejecución de la demo. Primero se encuentran las gráficas correspondientes a las fuentes de información:

Paquetes generados: paquetes generados de un tipo de fuente de información a lo largo del tiempo.

Kbytes enviados: kilo bytes generados de un tipo de fuente de información a lo largo del tiempo, separados por protocolo. Hay gráficas en las que no se encuentran todos los protocolos debido a que no se genera información en dicho protocolo.

Protocolos: porcentaje de protocolos a los que pertenecen los paquetes generados de un tipo de fuente de información a lo largo del tiempo.

BROWSING

Figura 13. Paquetes generados

Llega a un máximo de 260 paquetes enviados.

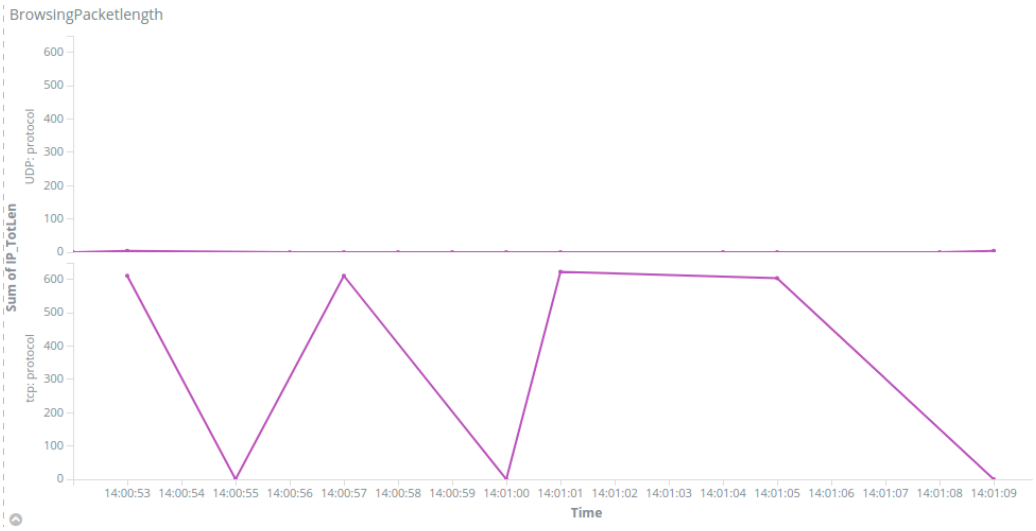

Figura 14. Kbytes enviados

Se envían una media de 400 kbytes por paquete en tcp mientras que, en UDP, en la parte inferior, apenas se genera información.

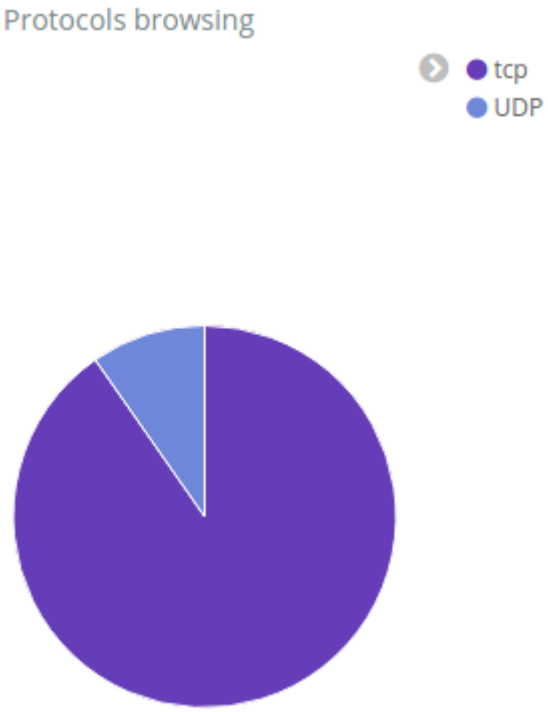

Figura 15. Protocolos

Como se apreciaba en la gráfica 14 y se corrobora con la figura 15, el mayor porcentaje de información se genera en el protocolo tcp.

VOIP

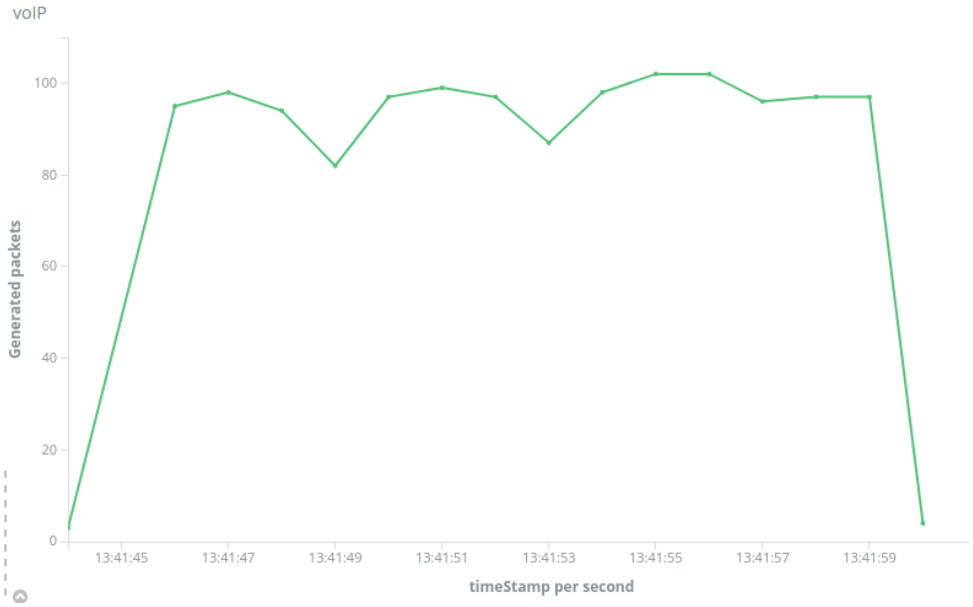

Figura 16. Paquetes generados

Llega a un máximo de 100 paquetes enviados.

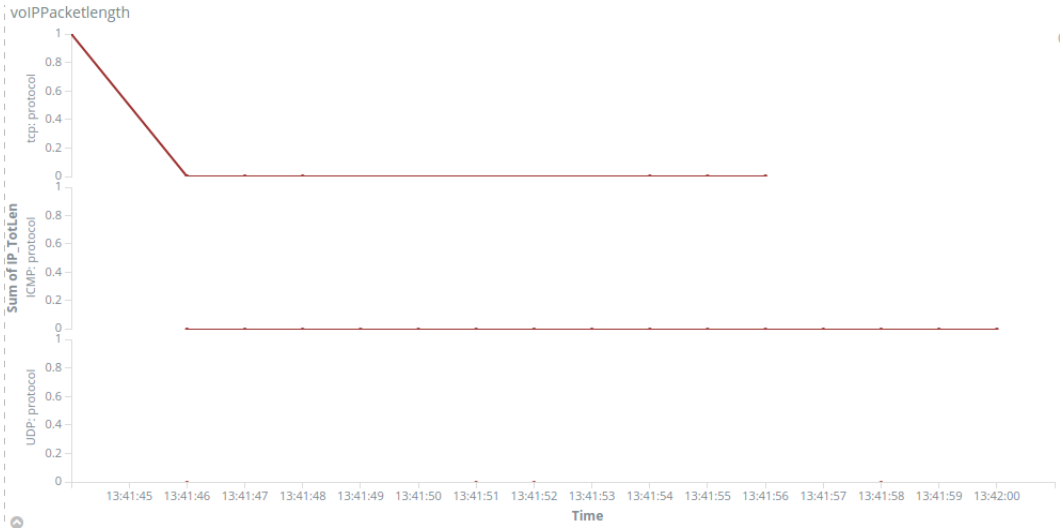

Figura 17. Kbytes enviados

En este caso podemos observar que se han generado en los 3 protocolos, pero el predominante ha sido el de TCP.

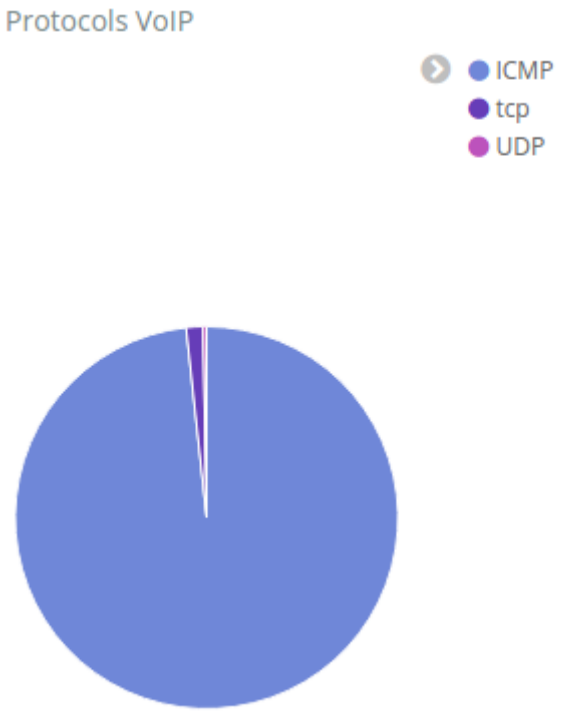

Figura 18. Protocolos

El procotolo TCP predomina frente a UDP e ICMP.

VIDEO

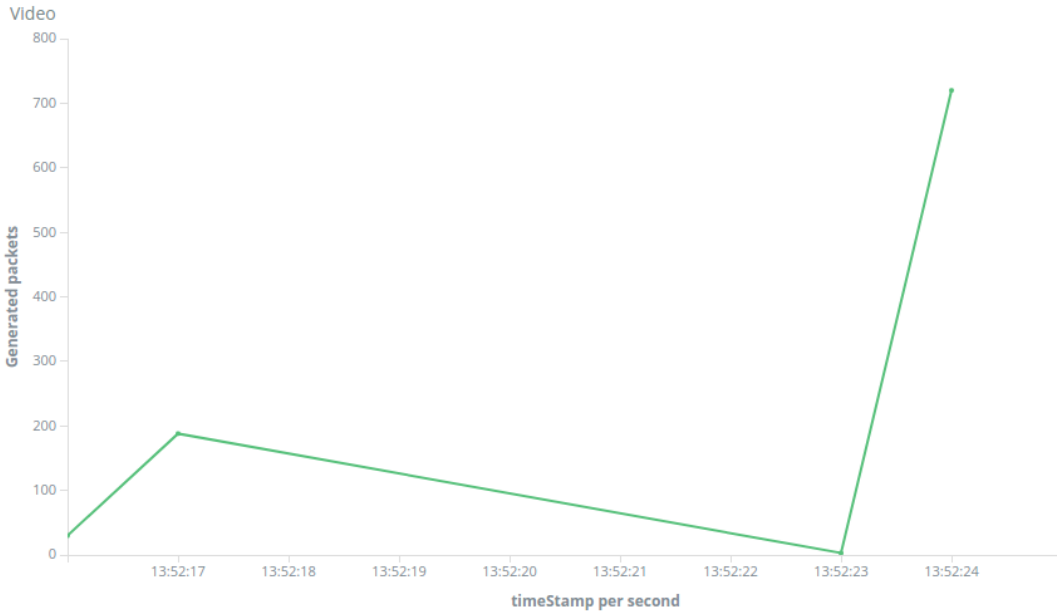

Figura 19. Paquetes generados

Llega a un máximo de 750 paquetes enviados.

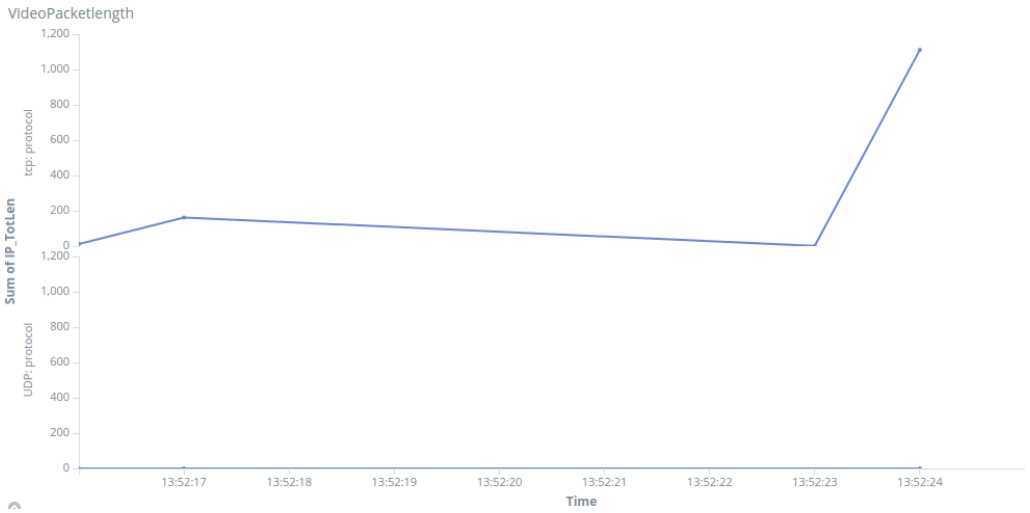

Figura 20. Kbytes enviados

Se llega a un máximo que supera los 1000 paquetes en la descarga de video. Predomina el protocolo TCP.

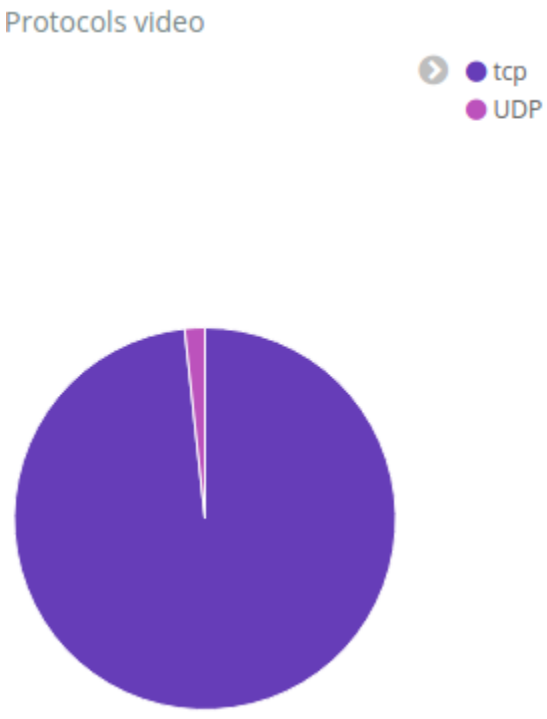

Figura 21. Protocolos

De nuevo en TCP vemos el protocolo predominante.

En segundo lugar se muestran dos gráficas pertenecientes a la evaluación del modelo a lo largo del tiempo.

MODELO

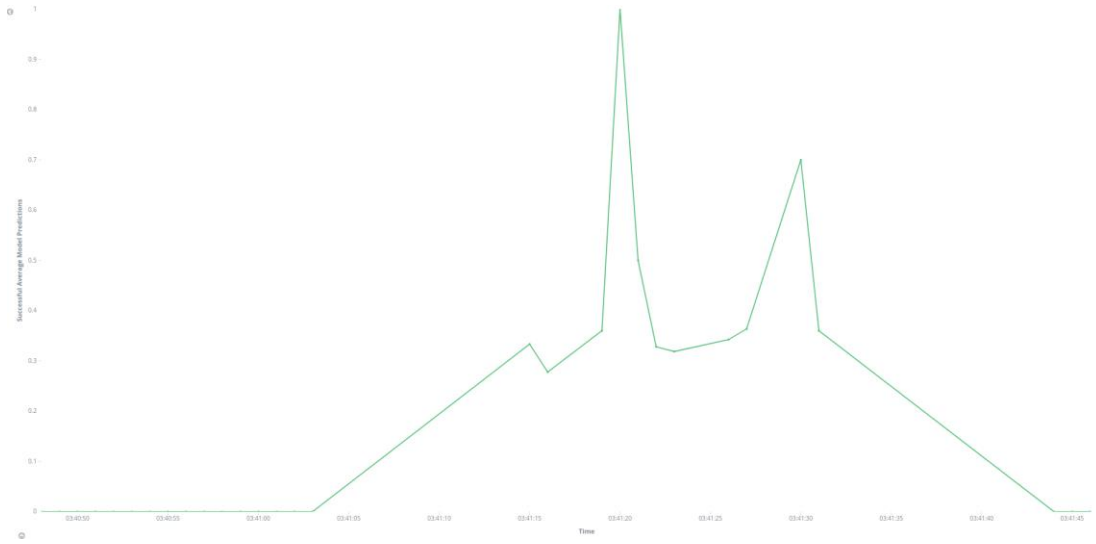

Figura 22. Porcentaje medio de aciertos

El período de tiempo seleccionado para la evaluación del modelo llega a un 60% de acierto en media. De máximo llega hasta casi un 100%.

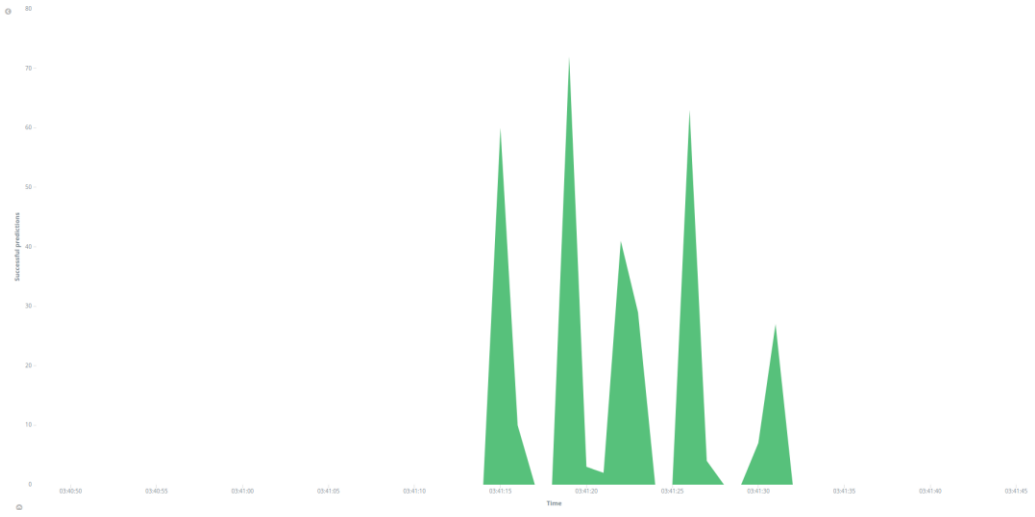

Figura 23. Número de aciertos

En la gráfica anterior lo que se observa es el número de aciertos del modelo. El pico más alto llega hasta los 70 paquetes.

Como nota general cabe señalar que los períodos en los que se miden los datos son de duración de 5 segundos.

5.2 Conclusiones finales

Una vez que llegamos a este punto, toca analizar y evaluar todo lo realizado.

En primer lugar, se puede confirmar que el objetivo perseguido, conseguir la visualización en *streaming* [20] de la evaluación de un modelo, se ha alcanzado y con buenos resultados.

Se ha llegado a conseguir una precisión de clasificación de casi el 80% en estático y una media de 65% en *streaming* [20] con el modelo Random Forest [13].

Por otro lado, se ha podido llevar a cabo la unión una gran selección de tecnologías simpatizantes con el análisis de grandes cantidades de datos.

En cuanto a objetivos secundarios, se ha profundizado en técnicas de machine learning aprendidas durante el Máster en Data Science y se ha puesto en consonancia con el uso de otras tecnologías.

5.3 Base y aprendizaje

Para el desarrollo del presente trabajo he utilizado conocimientos adquiridos durante la realización del Máster en Data Science.

Durante el desarrollo del proyecto he tenido que combinar los conocimientos adquiridos del tutor con los adquiridos de forma autodidacta.

5.3.1 Conocimientos previos

El primero de los conocimientos de los que partía de base es el lenguaje de programación utilizado para el desarrollo de la aplicación; Python. Su aprendizaje vino tanto de la carrera en Ingeniería en Sistemas de Telecomunicación y lo conocido en el Máster.

Además del máster, durante el periodo de trabajo que he llevado a cabo en la compañía Ericsson de telecomunicaciones en España, también he adquirido conocimientos acerca del lenguaje, técnicas de machine learning y sobre tráfico de red. De ahí la elección de un proyecto en el que se incluyeran las predicciones y el tráfico de red.

No se puede dejar sin mencionar el aprendizaje tanto en la universidad como en el trabajo acerca de las tecnologías Docker y Kafka.

5.3.2 Conocimientos adquiridos

El primer tema sobre el que tuve que investigar fue cómo poder modificar la salid del analizador TCPDUMP [16] para poder añadir campos de utilidad en la información generada en cada paquete. Finalmente lo llevé a cabo con Python [4].

El siguiente paso fue la integración de información generada en streaming con Kafka [7], la base de datos ElasticSearch [8] y la visualización de Kibana [9] . Con la incorporación de módulos de Python [4] entre medias de cada módulo pudo llevarse a cabo la comunicación entre tecnologías.

Por último, he tenido que profundizar en los métodos de predicción y machine learning que proporcionaba Spark [10] con las librerías Spark MLlib [11] y Spark ML [12], así como su integración con el *streaming* [20].

5.4 Líneas futuras

Para el desarrollo de esta aplicación se han asumido unas condiciones de entorno específicas, como la existencia de un solo usuario y 3 tipos específicos de fuentes de datos. Con estas condiciones se pretendía dar un comienzo a una aplicación muy potente si se guía por buen camino.

La primera mejora que se debería introducir en el programa es el cambio en la introducción de datos en *streaming* [20] en la base de datos. Actualmente se realiza con un cliente Python [4] pero podría llevarse a cabo de forma más efectiva con la tecnología *logstash*.

La siguiente mejora a incorporar sería la comparación de otros modelos que pudieran ajustarse mejor a las necesidades de los datos. Esto podría llevar a una mejor predicción de cada paquete. También sería interesante incorporar otras características de cada paquete para incluirlas en la creación del modelo.

Como se ha mencionado con anterioridad, en los papers [1] [2] leídos se realiza una clasificación de tráfico teniendo en cuenta flows enteros, no sólo paquetes. Este sin duda sería un salto importante a realizar, ya que se ajusta más al mundo real.

Una vez mejorada la estructura, se debería mejorar el modelo pudiendo discernir en la evaluación entre más de un usuario. Recordemos que actualmente sólo se considera un usuario.

Como conclusión final los resultados finales han sido satisfactorios para este primer prototipo de sistema, aunque aún queda trabajo que hacer para poder llegar a ser un sistema generalizado.

BIBLIOGRAFÍA

[1] "Analyzing Android Encrypted Network Traffic to Identify User Actions" Mario Conti, Luigi Vincenzo Mancini, Riccardo Spolaor, Nino Vincenzo Verde. IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 11, NO. 1, JANUARY 2016

http://ieeexplore.ieee.org/document/7265055/

[2] "Encrypted Traffic Analytics" Cisco, 2017.

https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf

[3] Ubuntu (versión 16.04)

https://www.ubuntu.com/

[4] Python (versiones 2.7 y 3.5)

https://www.python.org/

[5] Shell Bash

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

[6] Docker

https://www.docker.com/

[7] Apache Kafka

https://kafka.apache.org/

[8] ElasticSearch

https://www.elastic.co/

[9] Kibana

https://www.elastic.co/products/kibana

[10] Spark

https://spark.apache.org/

[11] Spark MLlib

https://spark.apache.org/mllib/

[12] Spark ML

https://spark.apache.org/docs/latest/ml-guide.html

[13] Random Forest

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

[14] Anaconda

https://docs.continuum.io/anaconda/

[15] Jupyter notebook

http://jupyter.org/

[16] TCPDUMP

http://www.tcpdump.org/tcpdump_man.html

[17] Definición Big Data

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

[18] Definición Ciencia de Datos (Data Science)

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

[19] Machine Learning

https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

[20] Streaming

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

[21] Spark Streaming

https://spark.apache.org/streaming/

[22] Logstash

https://www.elastic.co/products/logstash

[23] Gradient Boosting

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

[24] Multilayer Perceptron

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

APÉNDICE A

CÓDIGO DEL PROGRAMA

URL: https://github.com/IABcb/TFM.git