UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Desarrollo de una solución de clasificación de tráfico de red utilizando técnicas de aprendizaje automático

Raúl Santos Lebrato

Julio, 2018

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Desarrollo de una solución de clasificación de tráfico de red utilizando técnicas de aprendizaje automático

Autor: Raúl Santos Lebrato

Tutor: Francisco Javier Rodríguez Pérez

Co-Tutor/es: Francisco Javier Carmona del Río

Tribunal Calificador

Presidente: <Nombre y Apellidos>

Secretario: <Nombre y Apellidos>

Vocal: <Nombre y Apellidos>

ÍNDICE GENERAL DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

La importancia de la clasificación del tráfico ha crecido recientemente debido al crecimiento de la complejidad de las aplicaciones que buscan protegerse contra firewalls y bloqueos. Los proveedores de servicios de Internet necesitan una solución para poder clasificar el tráfico y ofrecer un mejor servicio. Las técnicas de clasificación también han evolucionado y, recientemente, la inteligencia artificial ha adquirido gran interés debido a que puede encontrar patrones difíciles de ver y en grandes cantidades de datos que una persona tardaría años en revisar manualmente, a parte de los problemas de privacidad que eso conllevaría. Por lo tanto, usamos las técnicas recientes de aprendizaje automático para comprobar si los algoritmos dan buenos resultados, cuál de ellos es más óptimo y cuál requiere un menor número de paquetes de entrenamiento para ofrecer una alta tasa de aciertos.

1. Introducción

A continuación se describe la situación actual (estado del arte) en clasificación de tráfico y las razones que llevan al desarrollo de este trabajo. Se explica por qué es útil poder clasificar tráfico, cómo se podría utilizar y por qué actualmente no hay un método preciso. Una vez explicada la situación y las motivaciones, se indican los objetivos del proyecto, más concretamente qué se pretende conseguir.

1.1. Antecedentes y contextualización

Los proveedores de servicios de Internet (ISP) dependen de herramientas de clasificación de tráfico para poder ofrecer un servicio justo y de calidad, necesitan saber qué tipo de tráfico circula por la red para poder optimizarla y dar mejor calidad de servicio o QoS (Quality of Service) [1][2]. Además, sabiendo a qué tipo de tráfico pertenecen los paquetes, los proveedores pueden aplicar mayor seguridad restringiendo qué tipos de tráfico pueden circular por la red.

Cada clase de tráfico puede ser tratada de forma distinta, por ejemplo dando preferencia al tráfico de servicios VoIP (Voice over Internet Protocol) o streaming, o intentar ofrecer un mejor encaminamiento de los datos [3]. Así como aplicando las distintas políticas de control, por ejemplo filtrando tráfico spam u otros ataques maliciosos.

En la actualidad, la clasificación del tráfico se lleva a cabo mediante el campo puerto del nivel de Transporte de TCP/IP (Transmission Control Protocol / Internet Protocol) ya que la IANA (Internet Assigned Numbers Authority) especifica 1024 números reservados para aplicaciones específicas y, de esta manera, se podría identificar la aplicación a partir del puerto que usa el paquete. HTTP (Hypertext Transfer Protocol) usa el puerto 80, HTTPS (HTTP Secure) usa el 443, Telnet usa el 23, etc.

Sin embargo, la IANA también permite usar un rango de puertos de forma libre para aplicaciones no especificadas. Esto hace el trabajo de clasificar el tráfico más difícil

si se trata de clasificar aplicaciones que no tienen un puerto fijo especificado, además, los programadores pueden especificar el puerto que quieran aunque se trate de una aplicación con un puerto estándar especificado por la IANA (se puede desplegar un servidor HTTP en un puerto que no sea el 80) [4].

Muchas aplicaciones usan estos puertos de libre uso o puertos que están registrados con la IANA para otras aplicaciones, con el objetivo de disfrazar el tráfico, también se usan puertos dinámicos o establecidos por el usuario e incluso se intenta ocultar el tráfico mediante el uso de túneles o encapsulado. De esta manera se intentan evitar los filtros o firewalls que bloquean tráfico de aplicaciones concretas [5].

1.2. Motivación

El número de aplicaciones que intentan ocultar su tráfico está creciendo debido a los incentivos para evitar el filtrado o bloqueo, por esta razón las técnicas de clasificación usadas hasta el momento, basadas en el puerto, dan resultados poco certeros [6]. Para solucionar este problema se han comenzado a estudiar nuevas técnicas de clasificación de tráfico [7], como el uso del contenido de los paquetes, es decir, el payload (DPI o Deep Packet Inspection) o algoritmos de inteligencia artificial.

Hay que tener en cuenta que el tráfico empieza a estar encriptado, cada vez más frecuentemente. Por lo que nos resultaría imposible clasificar este tipo de tráfico si nos basamos en el contenido del paquete, en estos casos las técnicas basadas en el uso del payload (DPI) dejan de funcionar, por lo que las técnicas basadas en algoritmos de inteligencia artificial, que usan información de la cabecera u otros datos estadísticos que puedan calcular, son la mejor opción [5][8].

La inteligencia artificial nos permite usar toda esta información que tenemos disponible acerca de cada paquete para intentar encontrar patrones entre los paquetes que pertenecen al mismo tipo de tráfico y, de esta manera, clasificarlo. Una tarea que sería prácticamente imposible de realizar manualmente.

1.3. Objetivos del proyecto

Este proyecto pretende analizar el tráfico que se produce en una red para poder tratar los paquetes de forma distinta según a qué tipo de tráfico pertenezcan. Para ello, se usarán varios algoritmos de Machine Learning, una rama de la inteligencia artificial cuyo objetivo es generalizar o encontrar patrones en los datos para poder predecir datos. Se comprobará la eficacia de esos algoritmos a la hora de clasificar los paquetes de un tráfico previamente capturado en varias clases predefinidas.

Es decir, habiendo capturado los paquetes que se envían y reciben en un uso normal de Internet con varias aplicaciones y habiendo etiquetado ese tráfico capturado, entrenamos a los algoritmos con parte de esos paquetes etiquetados. Machine Learning nos permite encontrar patrones de forma que al darle el resto de paquetes sin mostrarle la etiqueta, puede intentar predecir a qué etiqueta pertenece a partir de los patrones que ha encontrado con los datos de entrenamiento. Finalmente, comparamos las predicciones que hayan hecho los algoritmos con las etiquetas reales.

De esta manera podremos concluir qué algoritmo ofrece mejores resultados con el menor número de paquetes para su entrenamiento.

2. Análisis previo

Antes de poder realizar el estudio se busca información en trabajos similares y relacionados. En esta sección se indica la información extraemos de otros trabajos que nos sirve para decidir los algoritmos que usamos, así como las clases y los parámetros. Finalmente, se exponen las posibles soluciones a la clasificación de tráfico y cómo han ido evolucionando a lo largo del tiempo y se indica cuál es la solución que se propone en este trabajo y cómo se ha planificado el proyecto.

2.1. Análisis inicial

La elección los parámetros o campos a usar con los algoritmos de Machine Learning, así como el establecimiento de las clases en las que se clasifica el tráfico se ha realizado teniendo en cuenta trabajos anteriores [3][6][9][10].

Las clases elegidas son: GAME, VIDEOCHAT, LIVESTREAM, MEDIA, WEB. Estas clases son las más comúnmente encontradas en trabajos relacionados [6][10] [11] y son lo suficientemente diferenciadas para que la clasificación no de problemas.

Los parámetros seleccionados son: **tamaño**, **protocolo**, **IP origen**, **IP destino**, **puerto origen**, **puerto destino** y **tiempo entre paquetes**. A la hora de elegir estos parámetros se debe tener en cuenta que para llevar a cabo el proceso de identificación de tráfico en el pasado se ha estado usando el puerto debido a que IANA ha reservado puertos para aplicaciones específicas. A pesar de que este campo por sí mismo ya no resulta tan efectivo, sigue siendo útil tenerlo en cuenta, así como las direcciones IP, es decir, el socket del nivel de transporte de TCP/IP.

Sin embargo, otros parámetros nos resultan más útiles, como el protocolo que, aunque tan solo nos indica si se trata de tráfico TCP (Transmission Control Protocol) o UDP (User Datagram Protocol), debido a que el tráfico en tiempo real es principalmente UDP resulta ser un campo bastante útil a la hora de diferenciar las

clases con tráfico en tiempo real como GAME o VIDEOCHAT del resto, que usa TCP.

El campo tamaño también puede resultar muy útil ya que hay clases cuyo tráfico contiene paquetes pequeños en gran cantidad (como en tráfico en tiempo real) mientras que otras clases tienen tráfico con paquetes de mayor tamaño y quizás en menor cantidad (como el tráfico WEB).

Finalmente, también se tiene en cuenta el tiempo entre paquetes aunque no consideramos que aporte mucha información. Este campo se ha calculado a partir de los tiempos de llegada de los paquetes de un mismo tráfico (paquetes con los mismos sockets origen y destino). El primer paquete del tráfico tiene este campo a cero pudiendo causar confusión en estos casos. Sin embargo, se tiene en cuenta ya que habrá tráfico que deba ser rápido (menor tiempo entre paquetes) como el tráfico en tiempo real y tráfico que no tenga que ser tan rápido (mayor tiempo entre paquetes) como el tráfico WEB.

2.2. Análisis de requisitos

Para la clasificación del tráfico capturado, este tráfico se preprocesa para extraer la información que usan los algoritmos de Machine Learning. Estos campos son **tamaño**, **protocolo**, **IP origen**, **IP destino**, **puerto origen**, **puerto destino** y **tiempo entre paquetes**.

Los campos **tamaño** y **tiempo entre paquetes** son numéricos, mientras que el resto son cadenas de texto tratadas como variables categóricas. Finalmente estos datos son etiquetados según una serie de clases predefinidas: GAME, VIDEOCHAT, LIVESTREAM, MEDIA, WEB.

Se entrenan los algoritmos de Machine Learning por lo que para tráfico etiquetado con una de las cinco clases, el algoritmo debe clasificarlo con la misma clase indicada en su etiqueta. Para ello, es necesario un número suficiente de datos para el entrenamiento.

El número necesario de paquetes para un buen entrenamiento es también una de las variables que se estudia en este proyecto, se comprobará si un algoritmo necesita un entrenamiento con más paquetes para poder clasificar el tráfico con mayor precisión y a partir de qué número de paquetes no sigue mejorando la clasificación.

Usamos el 80% de los paquetes para entrenamiento y el 20% para las pruebas. Los paquetes son barajados y elegidos aleatoriamente para evitar que el estudio se vea afectado por el orden de los paquetes o un conjunto específico de ellos. Además, es necesario tener disponible un gran número de paquetes de tráfico lo suficientemente diferenciado para evitar que se den problemas de sobreajuste. Todo el tráfico que se usa en el proyecto es previamente capturado con la herramienta Wireshark intentado filtrar el ruido producido por aplicaciones en segundo plano.

Con la suficiente cantidad de tráfico este estudio debería poder proporcionar una buena clasificación de tráfico, por lo que podremos llegar a concluir qué algoritmo lo hace con mayor precisión.

2.3. Soluciones alternativas

Las técnicas de clasificación de tráfico han ido evolucionando con el desarrollo de protocolos y aplicaciones de Internet. Las aplicaciones son cada vez más sofisticadas y el crecimiento de los incentivos para ocultar el tráfico de forma que se puedan evitar filtros o bloqueos han hecho que las técnicas tradicionales (basadas en el puerto del nivel de Transporte) queden obsoletas [1]. Para solucionar este problema se han propuesto varias posibles técnicas, podemos agrupar las técnicas existentes de clasificación de tráfico en los siguientes grupos:

- **Técnicas basadas en el puerto:**

  Debido a que las aplicaciones actualmente usan números de puertos aleatorios, estas técnicas han perdido su efectividad. Eran las técnicas usadas tradicionalmente [1][5].

- **Técnicas basadas en el payload (DPI):**

Ya que las técnicas basadas en el puerto dejaron de ser eficaces, surgieron técnicas basadas en la inspección de los paquetes. Estas técnicas resuelven el problema de los puertos aleatorios, sin embargo, tienen otros problemas: Las políticas de privacidad o leyes pueden no permitir el acceso o almacenamiento del contenido de los paquetes, además, es una técnica que puede ser evitada fácilmente haciendo uso de encriptado o encapsulado [1] [5].

- **Técnicas basadas en el flujo:**

  Estas técnicas usan las estadísticas extraídas del flujo (tiempo de llegada entre paquetes, número total de paquetes, etc.) y evitan el uso del payload por lo que no les afectan los problemas de las técnicas basadas en él. Con aplicaciones cada vez más sofisticadas que tienden a usar varios protocolos al mismo tiempo estas técnicas solo pueden identificar protocolos individualmente, no aplicaciones enteras [1][4][6].

- **Técnicas basadas en el host:**

  Se intenta clasificar el tráfico monitorizando el tráfico enviado o recibido desde el mismo host, pero pierde efectividad cuando el host usa más de una aplicación [1][5][6].

- **Técnicas basadas en grafos:**

  Usa la relación entre los hosts ejecutando los mismos protocolos o aplicaciones, pero no está lo suficientemente desarrollada [1].

La figura 1 muestra cómo han ido evolucionando las técnicas de clasificación con la evolución de los protocolos y las aplicaciones.

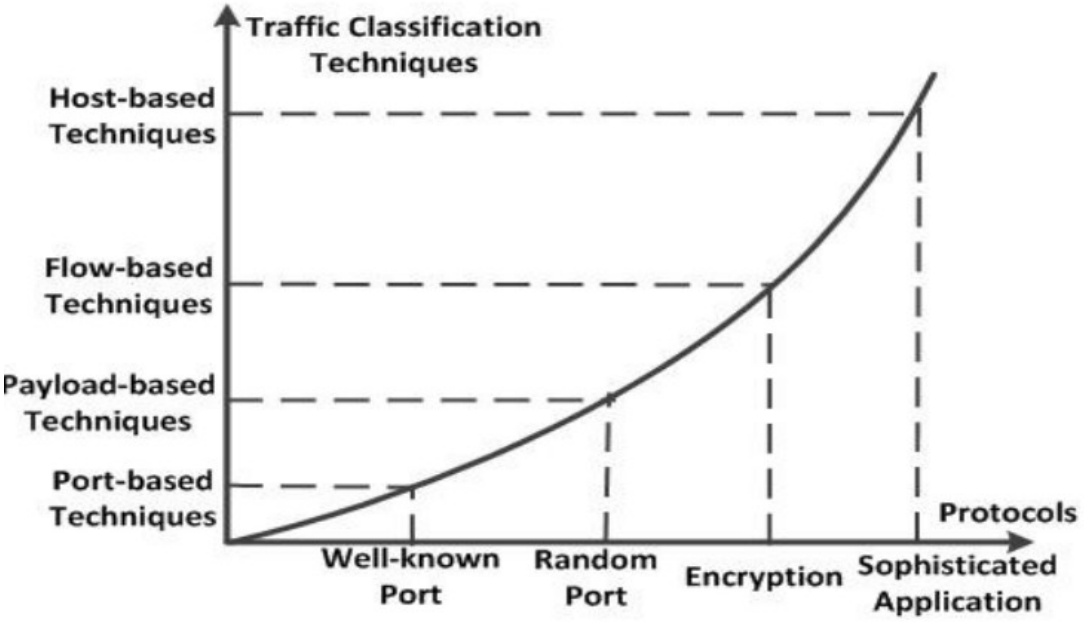

Figura 1: Evolución de protocolos y técnicas de clasificación de tráfico. Adaptada de [1], p. 241.

Existen varios estudios que hacen uso de algoritmos de Machine Learning para llevar a cabo la clasificación del tráfico, estos estudios usan parámetros estadísticos obtenidos del flujo por lo que se pueden considerar técnicas basadas en el flujo. El rendimiento de los algoritmos depende de las diferencias entre los algoritmos elegidos y su configuración específica así como de los parámetros o características elegidos para la clasificación [5].

Algunos de los algoritmos usados en estudios anteriores incluyen **C4.5** [6], **C5.0** [10], **Support Vector Machine** (SVM) [3][4] y **Naive Bayes** [2][3], encontrando que los algoritmos basados en árboles son los que mejores resultados suelen dar [8].

2.4. Solución propuesta

Una vez analizadas las distintas alternativas para dar respuesta a los objetivos planteados en este proyecto, se ha decidido utilizar Machine Learning para clasificar los paquetes del tráfico de la red en grupos predefinidos. Hemos elegido esta técnica ya que las alternativas no dan buenos resultados con aplicaciones sofisticadas y creemos que usando Machine Learning (que es ideal para encontrar patrones) se puede clasificar el tráfico de manera más precisa sobre todo con aplicaciones complejas.

Para el conjunto de datos (o dataset) de pruebas, consideramos preferible el realizar capturas reales de cada tipo de tráfico en lugar de seleccionarlos de una base de datos abierta donde se pueden obtener trazas de tráfico. El hecho de generar nuestras propias trazas correspondientes a cada tipo de tráfico, facilita la labor de identificación, que luego servirá para el entrenamiento y la validación. Este entorno de trabajo más controlado facilita el trabajo de preprocesado y nos da más control sobre la bondad de los resultados obtenidos.

La obtención del tráfico se realizará con la herramienta Wireshark. Esta herramienta nos permite exportar el tráfico capturado que luego preprocesaremos para extraer la información útil de cada paquete (los campos elegidos que se usarán para la clasificación) y etiquetarlos. Esta información se guardará en un archivo CSV (Comma-Separated Values) para que un script pueda leer los datos de manera eficiente (ya que las librerías que usaremos están diseñadas para trabajar de esta forma).

Una vez está preparado el dataset, podremos usarlo con los algoritmos de Machine Learning. Estos algoritmos usarán parte del tráfico ya etiquetado para entrenamiento (el 80%) y el resto para comprobar la tasa de acierto (el 20%), la división será siempre 80-20 ya que es la forma más común de dividir los datasets y se conoce como el principio de Pareto [12].

Tras realizar el entrenamiento, comprobaremos la eficacia de los algoritmos elegidos de Machine Learning: **Support Vector Machine** (SVM), **Decision Tree** y **Random Forest**. Estos algoritmos se han elegido ya que en el estudio previo son los más usados en trabajos similares o los que ofrecen mejores resultados. Los tres algoritmos son supervisados. La diferencia entre los algoritmos supervisados y los no-supervisados es que en los supervisados se usa un dataset ya clasificado para el entrenamiento de forma que el algoritmo pueda realizar una identificación y clasificación, mientras que en los no-supervisados solo se clasifica, es decir, agruparía los paquetes en varios grupos o clústeres intentando extraer un patrón de los datos. Aunque inicialmente consideramos incluir el algoritmo no-supervisado **k-Means** en el estudio, los resultados no nos resulta útil agrupar los paquetes de esta manera sin saber a qué tipo de tráfico pertenece cada clase por lo que solo se usan algoritmos supervisados en este estudio.

Los algoritmos elegidos se explican con más detalle en la sección de implementación [4.3. Implementación de la solución].

Finalmente, para comprobar la eficacia se compara la clase predicha por los algoritmos con la clase indicada en el campo etiqueta del dataset. De esta manera podremos comprobar qué algoritmo obtiene mejores resultados.

2.5. Planificación del trabajo

Para planificar el proyecto se ha utilizado un Diagrama de Gantt. Esta herramienta nos permite ver gráficamente el tiempo que se dedicará a cada tarea.

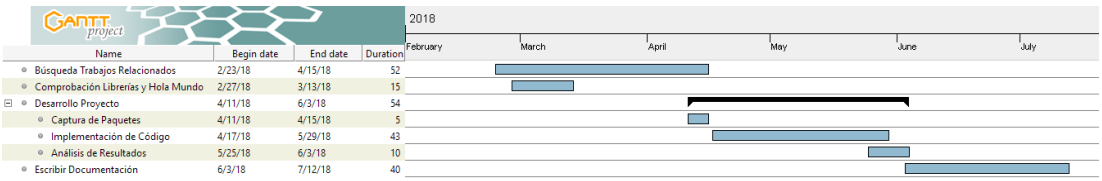

Figura 2: Diagrama de Gantt

Como se puede observar en la figura 2, el ciclo de vida del proyecto se ha dividido en varias tareas: **Búsqueda de Trabajos Relacionados**, **Comprobación de**

Librerías y Hola Mundo, **Desarrollo del Proyecto** y **Escribir Documentación**. Además, el desarrollo del proyecto se divide, a su vez, en 3 tareas: **Captura de Paquetes**, **Implementación de Código** y **Análisis de Resultados**.

En la tabla 1 podemos ver las fechas de inicio y terminación, así como la duración de estas tareas.

Tarea	Fecha de Inicio	Fecha de Terminación	Duración
Búsqueda de Trabajos Relacionados	23/02/2018	15/04/2018	52
Comprobación de Librerías y Hola Mundo	27/02/2018	13/03/2018	15
Desarrollo del Proyecto	11/04/2018	03/06/2018	54
Captura de Paquetes	11/04/2018	15/04/2018	5
Implementación de Código	11/04/2018	29/05/2018	43
Análisis de Resultados	25/05/2018	03/06/2018	10
Escribir Documentación	03/06/2018	12/07/2018	40

Tabla 1: Tareas del Ciclo de Vida del Proyecto.

3. Antecedentes y trabajo relacionado

Procedemos a mostrar las herramientas o librerías que se han considerado para realizar el trabajo e indicamos cuál ha sido elegida y por qué. Además, mostramos una tabla que resume la información extraída de trabajos relacionados como los algoritmos que usan, los parámetros y grupos de clasificación.

3.1. Herramientas de captura de paquetes

3.1.1. Scapy

Scapy es una librería para manipulación de paquetes, permite construir y decodificar paquetes de una gran variedad de protocolos, así como enviarlos o capturarlos [13].

Está desarrollada en Python y es la librería más conocida para captura de paquetes en Python.

Puede resultar una herramienta bastante limitada ya que ofrece una interfaz a alto nivel construida sobre libpcap. Sin embargo, es más que suficiente para el objetivo de este trabajo. Además, el hecho de que sea alto nivel ofrece gran facilidad de uso y velocidad de prototipado.

La gran facilidad de uso es una de las razones de su uso tan extendido en proyectos de investigación.

3.1.2. Libpcap

Libpcap es una librería de código abierto para la captura del tráfico de las redes [14]. Está desarrollada en C pero hay una gran multitud de wrappers creados por la comunidad para los lenguajes más usados (Python, Java, C#, Go…).

Esta librería es la más conocida y la más usada, una gran cantidad de programas y herramientas la usan (McAfee, Wireshark, tcpdump) y otras librerías a alto nivel se basan en ella (Scapy). Fue desarrollada por The Tcpdump team.

Está enfocada en la captura de paquetes a bajo nivel, fue originalmente desarrollada como parte de tcpdump y luego extraída como librería a parte.

3.1.3. Kaitai Struct

Una librería de código abierto desarrollada principalmente para parsear estructuras binarias [15].

Está desarrollada para poder usarse en una gran variedad de lenguajes (C++, C#, Go, Java, Javascript, Python…).

Debido a que esta librería se centra simplemente en el parseado de las estructuras, tendríamos que definir dichas estructuras manualmente, además de usar otra herramienta para la captura de los paquetes. Considero que esta herramienta está a demasiado bajo nivel y hay alternativas mejores.

3.1.4. Justificación Scapy

La herramienta elegida es Scapy. Libpcap es una librería de bajo nivel, mientras que Scapy es de alto nivel y usa libpcap por debajo. Es preferible usar una herramienta fácil de usar que permita un prototipado rápido. Además, a parte de permitir la captura de paquetes también permite su manipulación, a diferencia de libpcap.

Por otro lado, el uso de Scapy está más extendido en investigación por lo que habrá muchos trabajos relacionados que lo usen en los que se puede basar este proyecto, y muchos ejemplos útiles para el aprendizaje [16].

3.2. Herramientas de Machine Learning

3.2.1. Tensorflow

Tensorflow es una librería de código abierto para el cálculo numérico usando grafos de flujo de datos. Los nodos del grafo representan las operaciones matemáticas, mientras que las aristas representan los arrays de datos multidimensionales (tensores) [17].

La librería fue creada por Google con el fin de investigar machine learning. Como está pensada para la investigación, es una librería a muy bajo nivel que permite realizar operaciones matemáticas complejas a partir de las cuales el programador tendrá que implementar la inteligencia artificial, por lo que requiere un conocimiento matemático elevado.

El núcleo está implementado en C para poder crear wrappers en otros lenguajes fácilmente, el lenguaje de scripting principal es Python debido a lo extendido que está en proyectos de investigación pero existen wrappers oficiales para otros lenguajes populares (C++, Java y Go), también existen wrappers mantenidos por la comunidad.

Principalmente, se abrió el código para que la comunidad de programadores de inteligencia artificial se pueda beneficiar y avance la investigación. Sin embargo, para llegar a un más amplio conjunto de desarrolladores, también se creó un wrapper de más alto nivel para aquellos programadores que no tengan tantos conocimientos de matemáticas también puedan participar. Este wrapper permitía usar Sci-Kit Learn y finalmente fue fusionado con Tensorflow. De esta manera, la librería ofrece varias maneras de trabajar según tus conocimientos matemáticos ofreciendo gran flexibilidad a bajo nivel y una interfaz más limitada a alto nivel para el desarrollo de modelos comúnmente utilizados en inteligencia artificial [18].

Hay muchas empresas que utilizan Tensorflow además de Google (la creadora): airbnb, Nvidia, Uber, Dropbox, ebay, Snapchat, Intel, Coca-cola, ZTE, Qualcomm, Twitter, ARM, Lenovo…

3.2.2. Scikit-Learn

Scikit-learn es una librería de código abierto enfocada en la implementación de machine learning en Python. Es una de las librerías más conocidas por su gran facilidad de uso [19].

Está desarrollada en Python, haciendo uso de librerías matemáticas de Python (NumPy, SciPy y matplotlib). Originalmente fue desarrollada por David Cournapeau

como proyecto de Google Summer of Code, más tarde tomó el control un grupo de desarrolladores de INRIA.

Su objetivo es el minado y análisis de datos, por lo tanto, ofrecen herramientas que facilitan este objetivo ocultando la matemática que hay detrás. Por este motivo, es muy sencilla de usar para el desarrollo de modelos comúnmente utilizados en inteligencia artificial, es una librería de alto nivel [20].

Hay muchas empresas que utilizan scikit-learn, a parte de INRIA: Spotify, Evernote, Booking.com…

3.2.3. Keras

Keras es una librería de código abierto para implementar redes neuronales. Permite implementar la parte de inteligencia artificial que se refiere a las redes neuronales específicamente, para ello ofrece una interfaz de alto nivel con todo lo necesario para agilizar el proceso de desarrollo. Para usar la librería se ejecuta sobre una librería de más bajo nivel (que contiene las funciones matemáticas), puede ejecutarse sobre Tensorflow, CNTK o Theano [21].

Fue desarrollada originalmente por François Chollet, ingeniero de Google, como parte de un proyecto de investigación (ONEIROS).

Está desarrollada en Python, está pensada como librería de alto nivel que se use directamente en este lenguaje. Se centra sobre todo en la facilidad de uso, pero también permite modularizar los modelos y extender la librería para añadir nuevos modelos por lo que es posible usarla para investigación.

Su desarrollo está respaldado por grandes compañías: Google, Microsoft, Nvidia y Amazon.

3.2.4. Caffe

Caffe es una librería de deep learning centrado únicamente en imágenes desarrollado por la universidad de Berkeley [22]. Al solo poder usarlo para imágenes no nos sirve

para nuestro caso. Sin embargo, Facebook ha desarrollado Caffe2, una librería más flexible que ya no solo trabaja con imágenes.

Caffe2 está enfocada en permitir implementar deep learning de manera sencilla, con varios algoritmos comunes ya implementados [23]. Ha sido diseñada con la flexibilidad y modularidad en mente para evitar las limitaciones de Caffe 1.0.

La librería está implementada en C++ y su interfaz permite usar Python y C++.

3.2.5. Theano

Theano es una librería de código abierto que permite definir, optimizar y evaluar expresiones matemáticas en las que entran en juego arrays multidimensionales de manera eficiente [24]. Desarrollada por la universidad de Montreal.

La librería está desarrollada en Python sobre la librería matemática NumPy y está enfocada en la matemática a bajo nivel más que en la inteligencia artificial por lo que los proyectos de investigación de inteligencia artificial la suelen usar por debajo de una librería de alto nivel como Keras.

Para nuestro caso, esta librería está a demasiado bajo nivel y no nos resulta útil usarla directamente, por lo que sería mejor usar un wrapper de alto nivel.

3.2.6. Torch

Una librería de código de abierto para machine learning y cómputo científico. Está implementada en C y usa LUA como lenguaje de scripting [25].

Tiene gran soporte por parte de Facebook.

Debido a que usa el lenguaje LUA, el cual desconozco, considero que hay otras alternativas más conocidas que son preferibles para este trabajo.

3.2.7 Justificación Scikit-Learn

El lenguaje a usar será Python, por lo tanto, debido a su extendido uso en investigación la mayoría de las librerías nos sirven. Se da preferencia a que sean librerías de alto nivel (fácil uso) ya que no soy un matemático y no tengo pensado

implementar un algoritmo de inteligencia artificial sino usar algoritmos ya existentes, las más fáciles son Scikit-Learn y Tensorflow.

Se han probado Tensorflow y Scikit-Learn para decidir cuál usar ya que ambos son muy parecidos. Ambos son sencillos de usar y nos permiten realizar el trabajo objetivo, en la prueba realizada el código relacionado con inteligencia artificial resultó ser prácticamente idéntico (los mismos 3 métodos con distintos nombres: (el constructor de la clase, el método train/fit y el método predict)), la mayoría del código es el relacionado con el preprocesamiento de los datos antes de ser tratados por el algoritmo de inteligencia artificial. El código implementado de prueba es un sencillo Hola Mundo con el dataset Iris [18].

La gran diferencia entre Scikit-Learn y Tensorflow quizás sea que Tensorflow se centra en Deep Learning, algoritmos de inteligencia artificial modernos, mientras que Scikit-Learn se centra en Machine Learning, algoritmos de inteligencia artificial clásica. Para este trabajo nos centraremos en algoritmos de Machine Learning debido a que ya dan resultados muy buenos y Deep Learning requeriría un volumen de datos mucho mayor para empezar a dar buenos resultados.

Por lo tanto y como este trabajo se centra en algoritmos de Machine Learning se ha decidido usar Scikit-Learn.

3.3. Trabajos relacionados

Criterio	Algoritmos	Nivel de clasificación	Observaciones/Conclusión a priori	Ref
Campos de la cabecera y tiempo	Naive Bayes (NB)	Grupos (BT, DNS, FTP, HTTP, IMAP, MSN, POP3, SMTP, SSH, SSL, XMPP)	Nuevo esquema de clasificación de tráfico que mejora el rendimiento cuando pocos datos de entrenamiento están disponibles.	[26]
Campos de la cabecera, paquetes y tiempo	Traffic Classification using Correlation (TCC)	Grupos (P2P, DNS, FTP, WWW, CHAT, MAIL) y (BT, DNS, eBuddy, FTP, HTTP, IMAP, MSN, POP3, RSP, SMTP, SSH, SSL, XMPP, YahooMsg)	TCC se implementa usando AVG-NN, MIN-NN y MVT-NN (en general AVG-NN da mejores resultados aunque depende de la cantidad de datos de entrenamiento). **Trabajo futuro:** Se centran en TCP y dejan tráfico no-TCP (UDP) para futuro trabajo.	[27]
Campos de la cabecera y Métricas de flujo (con menos importancia)	Bayesian Network (BN), C4.5 y Multilayer Perceptron	Grupos (P2P, Content Delivery, Web (HTTP), Bulk (FTP), Service (DNS), Mail (SMTP))	**Trabajo futuro:** Rendimiento con algoritmos ML unsupervised. La clasificación de tráfico P2P. Efectos de incluir diferentes aplicaciones individuales P2P en dataset.	[9]
Campos de la cabecera, tiempo y Métricas de flujo	Support Vector Machines (SVM)	Grupos (Bulk (FTP), Interactive (SSH), Mail (POP3, SMTP), Service (DNS), WWW (HTTP), P2P, Multimedia, Game, Attack, Others)	Alta precisión (99+%) usando parámetros de flujo obtenidos de la cabecera solo. **Trabajo futuro:** SVM requiere un alto número de datos etiquetados e identificar el tráfico tras obtener el flujo podría ser demasiado tarde. Se intentará combinar algoritmos ML supervised + unsupervised y usar parámetros obtenibles temprano en el flujo.	[4]
Campos de la cabecera y Métricas de flujo	NB, k-Nearest Neighbors (k-NN), C4.5 y SVM	Grupos (Web, P2P, FTP, DNS, Mail/News, Streaming, Network Operation, Encryption, Games, Attach, Unknown)	Precisión disminuye si se pierden los primeros paquetes del flujo. Aun la más pequeña discretización (como Equal-Interval Width en tamaño de paquetes) mejora rendimiento, no tanto como métodos basados en la entropía. **Trabajo futuro:** Clasificación de tráfico UDP ya que usan métricas de conexión TCP bidireccional.	[6]
Métricas de flujo y características dependientes del protocolo	C5.0	Grupos (Skype, FTP, Torrent, Web, Web radio, Game, SSH)	Mejores resultados combinando métricas con atributos de puerto y cuando las métricas se obtienen de un gran número de paquetes y no solo unos pocos, máximo 35. **Trabajo futuro:** usar datos obtenidos de gran cantidad de usuarios, no solo unos pocos, y en distintas redes.	[11]

Criterio	Algoritmos	Nivel de clasificación	Observaciones/Conclusión a priori	Ref
Métricas de flujo	Naive Bayes (NB)	Grupos (Bulk (FTP), Database, Interactive (SSH, TELNET), Mail (POP3, SMTP), Services (DNS), WWW, P2P, Attack, Games, Multimedia)	**Trabajo futuro:** Probar spatial-independence. Refinar discriminadores, de esto se beneficiaría el tráfico P2P.	[2]
Métricas de flujo	Perceptron Network	Grupos (Bulk, Database, Interactive, Mail, Services, WWW, P2P, Attack, Games, Multimedia)	Diferencias "spatial" tienen poca relevancia en Naive Bayes y C4.5 (deben probarse mayores diferencias). Con separación temporal todos los sistemas tienen peor precisión, en este caso la red neuronal es la que mejor resultados obtiene. C4.5 parece mejor para clasificación de tráfico. **Trabajo futuro:** Probar con datos adicionales y otros métodos de optimización de la red neuronal. Intentar identificar clases específicas.	[11]
Campos de la cabecera y tiempo	NB, Logistic Regression, SVM, k-Means clustering	Streaming Video u Otro	SVM fue el mejor.	[3]
Métricas de flujo	Random Forest, Gradient Boosted Trees	Video o Non-Video	**Trabajo futuro:** Cross-dataset evaluation (más datos, no solo de cellular-data), feature engineering (ver si otras características dan mejor resultado), mejor etiquetado, más clases (diferentes características podrían funcionar mejor según la clase).	[8]

Tabla 2: Resumen de la información extraída de trabajos relacionados.

El tráfico, en todos los casos, es generado a partir de usuarios haciendo uso normal de un computador por lo que es completamente aleatorio y se recoge tráfico de varias aplicaciones.

En la columna **criterio** se indican los parámetros usados para la clasificación, en la mayoría de los casos se usan campos de la cabecera (como el puerto), también se suelen usar métricas de flujo (parámetros calculados a partir de un flujo de paquetes), también se usan en algunos trabajos el tiempo (esto es el tiempo transcurrido entre paquetes pertenecientes al mismo flujo) o características dependiente de protocolo (campos que dependen de si el protocolo es TCP o UDP). Cabe destacar que en uno de los trabajos [9] se comprobó que al usar campos directamente extraídos de la cabecera y campos calculados a partir del flujo, estos últimos tenían menos importancia a la hora de clasificar, es decir, no mejoraba los resultados considerablemente.

En la columna **algoritmos** se destacan los algoritmos de los que se habla en el trabajo referenciado, algunos se centran en un solo algoritmo mientras que otros comparan los resultados entre varios algoritmos.

En la columna **nivel de clasificación** se especifican las categorías o clases que se usaron para la clasificación, la mayoría de los trabajos usan una multitud de grupos genéricos muy similares, pero hay dos trabajos [3][8] que simplemente clasifican el tráfico en dos clases.

La columna **observaciones/conclusión a priori** pretende destacar observaciones importantes de los trabajos que se obtuvieron al principio del desarrollo de este estudio así como posible trabajo futuro mencionado.

Finalmente, la columna **ref** indica a qué trabajo hace referencia la fila de la tabla.

4. Desarrollo de la solución

A continuación se explica la solución que se desarrolla en este trabajo. Se indican los casos para los que es útil y por qué hacer la clasificación con Machine Learning. Se explica el sistema completo que se puede dividir en tres partes: **captura**, **preprocesado** y **clasificación** del tráfico. Y finalmente se explica la implementación de la solución, describiendo los algoritmos que se utilizan (**SVM**, **Decision Tree** y **Random Forest**) y cómo funcionan, así como ejemplos de uso del proyecto.

4.1. Casos de uso

Este proyecto comprueba si es posible clasificar el tráfico de manera precisa con algoritmos de Machine Learning y cuál de ellos otorga mejores resultados. Esta información es útil para los proveedores de servicios de Internet para que puedan clasificar el tráfico con mayor precisión y, de esta manera, ofrecer un mejor servicio.

La clasificación del tráfico de una red resulta útil para realizar la administración y el diseño de redes, así como para monitorizar su seguridad, permitir su contabilidad y asegurar la calidad del servicio (QoS) [1][2][4][5]. Además, al poder identificar el tráfico, los proveedores pueden ofrecer un servicio de Internet fiable y justo, pudiendo restringir tráfico específico o implementando mecanismos para la diferenciación del servicio que permita dar preferencias a tipos específicos de tráfico que la necesiten [3][5].

4.2. Descripción del sistema completo

El sistema completo se compone de capturas de tráfico así como scripts que tratan estas capturas y ejecutan algoritmos de Machine Learning para clasificarlo.

Podemos dividir el proceso en tres partes: **captura del tráfico**, el **preprocesado del tráfico** y la **clasificación del tráfico**. El preprocesado se lleva a cabo en 2 pasos: **generación del CSV** y **división por número de paquetes**. Todo este proceso se ha representado en la figura 3.

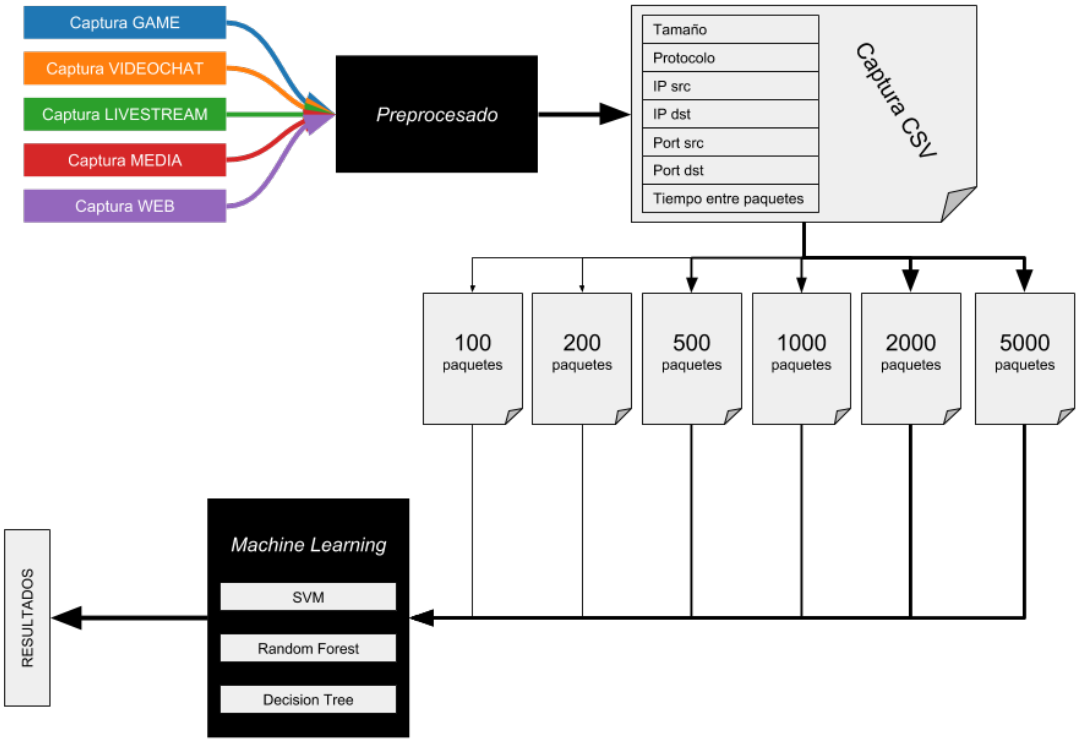

Figura 3: Esquema del sistema completo

El sistema completo está compuesto por 3 scripts Python: **pcap_to_csv.py**, **reduce_capture.py** y **main.py**. Los dos primeros se encargan del preprocesado, primero extraemos los datos de la captura de Wireshark en un solo CSV y después generamos varios CSV con distintos tamaños (100 paquetes por clase, 200 paquetes por clase, etc.), el último script (main.py) se encarga de entrenar y comprobar la

precisión de los algoritmos de Machine Learning a partir de los varios ficheros CSV y generar los gráficos con los resultados.

4.2.1. Captura del tráfico

Para la ejecución de los algoritmos, necesitamos obtener datos que puedan tratar, para ello se ha usado la herramienta Wireshark para capturar tráfico perteneciente a las clases predefinidas en el análisis inicial: GAME, VIDEOCHAT, LIVESTREAM, MEDIA, WEB.

Clase	Descripción
GAME	Tráfico de un videojuego multijugador (Una partida en Counter Strike, juego FPS).
VIDEOCHAT	Tráfico de una video-llamada de 15 minutos usando la aplicación Skype en Windows 10.
LIVESTREAM	Tráfico de la retransmisión de un live streaming en YouTube usando la aplicación OBS en Windows 10.
MEDIA	Tráfico perteneciente a la reproducción de dos vídeos de 3 minutos y uno de 10 minutos usando la página YouTube en Chrome en Windows 10.
WEB	Tráfico de navegación por varias páginas web (Google, DuckDuckGo, NYTimes, Washington Post, BBC, CNN…) usando Chrome en Windows 10.

Tabla 3: Clases o categorías de aplicaciones en las que se clasificará el tráfico.

El tráfico generado es lo más diferenciado posible. Además, se han filtrado las capturas para eliminar posible ruido generado por aplicaciones ejecutándose en segundo plano en el momento de la captura.

4.2.2. Preprocesado del tráfico

El script que se encarga de esta parte del proceso leerá las capturas realizadas con la herramienta Wireshark y se encargará de extraer la información que usarán los algoritmos de Machine Learning, esta información son los campos elegidos en el análisis inicial: **tamaño**, **protocolo**, **IP origen**, **IP destino**, **puerto origen**, **puerto destino** y **tiempo entre paquetes**.

Parámetro	Descripción
Tamaño	Tamaño en bytes del paquete.
Protocolo	Protocolo del Nivel de Transporte usado (TCP o UDP).
IP src	Dirección IP origen del paquete.
IP dst	Dirección IP destino del paquete.
Port src	Puerto origen asignado a la aplicación que lo envía.
Port dst	Puerto destino asignado a la aplicación que debe recibirlo.
Tiempo entre Paquetes	Tiempo transcurrido entre el paquete actual y el anterior perteneciente al mismo socket (si no hay anterior será 0).
Label	Etiqueta o clase que identifica el tráfico (será una de las clases definidas previamente). Será usado para entrenamiento o comprobación de la tasa de acierto.

Tabla 4: Parámetros o características usadas para la clasificación.

El etiquetado de los paquetes se lleva a cabo según a qué clase pertenece cada captura realizada y será utilizada en el entrenamiento como salida, así como a la hora de probar la predicción para comprobar que las clases elegidas por los algoritmos son correctas.

4.2.3. Clasificación del tráfico

Finalmente teniendo los datos extraídos, los algoritmos de Machine Learning leerán estos datos y usarán el 80% elegidos aleatoriamente para el entrenamiento, y el 20% restante para comprobar la tasa de acierto.

4.3. Implementación de la solución

Se usará Machine Learning para clasificar los paquetes del tráfico de la red en grupos predefinidos. Para ello capturaremos previamente tráfico perteneciente a varias aplicaciones y lo etiquetaremos para usarlo con los algoritmos de Machine Learning. Estos algoritmos usarán parte del tráfico ya clasificado para entrenamiento y el resto para comprobar la tasa de acierto.

El tráfico previamente capturado pertenece a 5 clases: GAME, VIDEOCHAT, LIVESTREAM, MEDIA, WEB.

Usando la librería Scapy se ha implementado un script que recorres las capturas realizadas con Wireshark y lo procesa para quedarnos solo con la información que nos interesa que use el algoritmo de Machine Learning, algunos de estos parámetros se obtienen directamente de la cabecera de cada paquete mientras que otros son calculados (por ejemplo, el tiempo transcurrido entre el paquete actual y el anterior recibidos que pertenecen al mismo socket).

Los parámetros que usaremos con los algoritmos de Machine Learning son los siguientes: **tamaño**, **protocolo**, **IP origen**, **IP destino**, **puerto origen**, **puerto destino** y **tiempo entre paquetes**.

Una vez hemos procesado los paquetes, esta información se guarda como archivos CSV ya que las librerías que usamos ya implementan métodos que nos permiten leer fácilmente este tipo de archivos. Probaremos la eficacia de los algoritmos con varias cantidades de paquetes por lo que los CSV a su vez se dividirán en otros más pequeños con un número n de paquetes de cada clase cogiendo los paquetes a partir

del centro de cada clase. Probaremos con 100, 200, 500, 1000, 2000 y 5000 paquetes por clase, debido a que la clase con menos paquetes (WEB) solo tiene 6860.

En otro script usamos la librería Scikit-Learn para leer los archivos CSV y ejecutar los algoritmos de Machine Learning sobre los datos. Usaremos el 80% de los paquetes para entrenamiento y el 20% para las pruebas. Una vez han sido entrenados los clasificadores y se han ejecutado las pruebas, el resultado es una matriz de confusión que nos permite ver cuántos paquetes ha clasificado correctamente y cuántos incorrectamente para cada clase.

Finalmente generamos las gráficas que representan los resultados.

4.3.1. Support Vector Machines

Support Vector Machines o máquinas de soporte vectorial son un conjunto de algoritmos de aprendizaje supervisado desarrollados por Vladimir Vapnik. Dado un conjunto de ejemplos o muestras de entrenamiento etiquetado en clases concretas, se construye un modelo SVM para predecir la clase de un nuevo ejemplo. SVM es un modelo que representa los puntos de la muestra en el espacio, separando las clases lo máximo posible [28].

Las ventajas de SVM son:

- Efectivos en espacios de alta dimensionalidad.
- Efectivo en casos donde el número de dimensiones es mayor al número de muestras.
- Usa un subconjunto de puntos de entrenamiento en la función de decisión, por lo que es eficiente con la memoria.
- Es versátil, se pueden especificar distintas funciones Kernel para la función de decisión.

Las desventajas de SVM son:

- Si el número de características es muy superior al número de muestras, es crucial evitar sobreajuste eligiendo una función Kernel adecuada y regularizando.
- No ofrecen estimaciones de probabilidad directamente, se pueden calcular usando la validación cruzada.

La clase SVC (C-Support Vector Classification) de scikit-learn es un clasificador que acepta múltiples clases usando el esquema one-vs-one [29].

```
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC

clf = SVC()
clf.fit(X_train, y_train)

y_predicted = clf.predict(X_test)
confusion_matrix(y_expected, y_predicted, labels)
```

El constructor SVC toma varios parámetros para configurar el clasificador:

- **C** (por defecto = 1.0)

  Parámetro de penalización C. Indica cuánto se quiere evitar una clasificación errónea por cada muestra de entrenamiento, con valores altos la optimización usará un margen menor en el hiperplano.

- **kernel** (por defecto = 'rbf')

  Especifica el tipo de kernel que usará el algoritmo. En nuestro caso usamos rbf (Radial Basis Function kernel) y se representa como:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

- **gamma** (por defecto = 'auto')

  Coeficiente del kernel, al ser auto se usará como valor 1/n_features donde n_features es el número de características.

- **libreprobability** (por defecto = False)

  Indica si se habilitan estimaciones de probabilidad, lo dejamos desactivado.

- **shrinking** (por defecto = True)

  Indica si se usa la heurística de reducción o "shrinking", lo dejamos activado.

- **tol** (por defecto = 1e-3)

  Tolerancia para el criterio de parada.

- **max_iter** (por defecto = -1)

  Número máximo de iteraciones, al ser -1 no habrá límite.

- **decision_function_shape** (por defecto = 'ovr')

  Indica si devolverá una función de decisión one-vs-rest (ovr) con forma (n_samples, n_classes) o bien one-vs-one (ovo) con forma (n_samples, n_classes * (n_classes - 1) / 2). Usamos la recomendada ovr.

- **random_state** (por defecto = None)

  La semilla del generador de números pseudoaleatorio que se usará para barajar los datos, al ser None usará la instancia RandomState usada por np.random.

4.3.2. Decision Tree

Decision Tree o Árbol de Decisión es un modelo de predicción que usa un modelo con forma de árbol o grafo donde se representan las decisiones y sus posibles consecuencias. Es una forma de representar un algoritmo que solo contiene sentencias de control. El objetivo de los árboles de decisión es crear un modelo que

predice la clase habiendo aprendido simples reglas de decisión a partir de los parámetros de los datos [30].

Las ventajas de los árboles de decisión son:

- Fáciles de entender e interpretar, pueden ser visualizados.
- El coste del uso del árbol es logarítmico.
- Puede manejar datos numéricos o categóricos.
- Puede manejar problemas con múltiples salidas.
- Si una situación es observable en un modelo, la explicación se puede explicar con lógica booleana.
- Buen rendimiento incluso cuando se hacen suposiciones que difieren del modelo verdadero.

Las desventajas de los árboles de decisión son:

- Pueden crear árboles excesivamente complejos que no generalicen bien los datos, es decir, se puede producir un sobreajuste.
- Pueden ser poco estables debido a pequeñas variaciones en los datos que resulten en generar un árbol completamente distinto.
- Generar un árbol que tome la decisión más óptima es un problema NP-completo. Los algoritmos de aprendizaje se basan en heurísticas como la voraz donde la decisión óptima se toma en cada nodo, por lo que no se puede garantizar que se tome una decisión óptima global. Esto se podría solucionar entrenando múltiples árboles donde las características y muestras son aleatorias (así es como funciona Random Forest).
- Hay conceptos que son difíciles de aprender porque los árboles de decisión no los expresan fácilmente, como XOR.

- Los árboles de decisión son sesgados si una de las clases aparece mucho más que las demás, por esto se recomienda balancear el dataset antes de realizar el ajuste.

En scikit-learn, la clase DecisionTreeClassifier es un clasificador capaz de realizar una clasificación multi-clase sobre un dataset [31].

```
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

y_predicted = clf.predict(X_test)
confusion_matrix(y_expected, y_predicted, labels)
```

El constructor DecisionTreeClassifier toma varios parámetros para configurar el clasificador:

- **criterion** (por defecto = 'gini')

  La función que medirá la calidad de un split o división. Tomará el valor 'gini' para usar la impuereza de Gini o 'entropy' para usar la ganancia de información. En nuestro caso usamos la impureza de Gini, para calcularla sobre un conjunto de elementos donde i = {1, 2, ..., m} y fi es la fracción de artículos etiquetados con valor i en el conjunto:

$$I_G(f) = -\sum_{i=1}^{m} f_i \log_2(f_i)$$

- **splitter** (por defecto = 'best')

  La estrategia usada para elegir el split o división en cada nodo. Puede ser 'best' para elegir la mejor división o 'random' para elegir la mejor división aleatoria, usamos 'best'.

- **max_depth** (por defecto = None)

  La máxima profundidad del árbol, al ser None los nodos se expandirán hasta que todas las hojas sean puras o contengan menos muestras que el valor de min_samples_split.

- **min_samples_split** (por defecto = 2)

  El número mínimo de muestras requeridas para dividir un nodo interno. En nuestro caso 2.

- **min_samples_leaf** (por defecto = 1)

  El número mínimo de muestras requeridas para estar en un nodo hoja. En nuestro caso 1.

- **min_weight_fraction_leaf** (por defecto = 0)

  La fracción mínima ponderada de la suma total de pesos requerida para estar en un nodo hoja. En nuestro caso no ponderamos las muestras.

- **max_features** (por defecto = None)

  El número de características a considerar cuando se busca la mejor división. Al ser None se usarán todas las características.

- **random_state** (por defecto = None)

  De la misma manera que en SVC, determina la semilla usada por el generador de números aleatorios y al ser None usará la instancia RandomState de np.random.

- **max_leaf_nodes** (por defecto = None)

  El número máximo de nodos hoja, al ser None no hay máximo por lo que habrá un número ilimitado de nodos hoja.

- **min_impurity_decrease** (por defecto = 0)

  Un nodo se dividirá si el split o división provoca un decremento de la impuridad mayor o igual al valor de este parámetro.

- **presort** (por defecto = False)

  Indica si se ordenarán los datos previamente. Lo dejamos desactivado.

4.3.3. Random Forest

Random Forest o Bosques Aleatorios es una combinación de árboles predictores en los que cada árbol depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de ellos, la predicción es la media de los clasificadores individuales. Fue desarrollado por Leo Breiman y Adele Cutler. Corrigen la tendencia a sobreajuste de los Decision Trees [32].

El clasificador de scikit-learn es un meta-estimador que ajusta un número de clasificadores del tipo Decision Tree en varios subconjuntos de ejemplos o muestras del dataset, haciendo la media para mejorar la precisión de la predicción y controlar el sobreajuste [33].

```
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100, oob_score=True)
clf.fit(X_train, y_train)

y_predicted = clf.predict(X_test)
confusion_matrix(y_expected, y_predicted, labels)
```

El constructor RandomForestClassifier toma varios parámetros para configurar el clasificador, como este algoritmo usa árboles de decisión tendrá muchos parámetros en común con el constructor de DecisionTreeClassifier:

- **n_estimators** (por defecto = 10)

  El número de árboles en el bosque. Nosotros usaremos 100 ya que, en general, los resultados son mejores cuanto mayor número de árboles, sin embargo, hay un número máximo de árboles a partir del cual empiezan a empeorar los resultados por lo que usar un número muy alto de árboles puede empeorar la tasa de acierto y hay que elegir un número adecuado.

- **criterion** (por defecto = 'gini')

  La función que medirá la calidad de un split o división. Tomará el valor 'gini' para usar la impuereza de Gini o 'entropy' para usar la ganancia de información. En nuestro caso usamos la impureza de Gini, para calcularla sobre un conjunto de elementos donde i = {1, 2, ..., m} y fi es la fracción de artículos etiquetados con valor i en el conjunto:

$$I_G(f) = -\sum_{i=1}^{m} f_i \log_2(f_i)$$

- **max_depth** (por defecto = None)

  La máxima profundidad del árbol, al ser None los nodos se expandirán hasta que todas las hojas sean puras o contengan menos muestras que el valor de min_samples_split.

- **min_samples_split** (por defecto = 2)

  El número mínimo de muestras requeridas para dividir un nodo interno. En nuestro caso 2.

- **min_samples_leaf** (por defecto = 1)

  El número mínimo de muestras requeridas para estar en un nodo hoja. En nuestro caso 1.

- **min_weight_fraction_leaf** (por defecto = 0)

  La fracción mínima ponderada de la suma total de pesos requerida para estar en un nodo hoja. En nuestro caso no ponderamos las muestras.

- **max_features** (por defecto = None)

  El número de características a considerar cuando se busca la mejor división. Al ser None se usarán todas las características.

- **random_state** (por defecto = None)

  De la misma manera que en SVC y DecisionTree, determina la semilla usada por el generador de números aleatorios y al ser None usará la instancia RandomState de np.random.

- **max_leaf_nodes** (por defecto = None)

  El número máximo de nodos hoja, al ser None no hay máximo por lo que habrá un número ilimitado de nodos hoja.

- **min_impurity_decrease** (por defecto = 0)

  Un nodo se dividirá si el split o división provoca un decremento de la impuridad mayor o igual al valor de este parámetro.

- **bootstrap** (por defecto = True)

  Por cada árbol en el Random Forest hará su propia división aleatoria de entrenamiento/pruebas, a este procedimiento se le conoce como agregación de bootstrap o empaquetado y las muestras que caen fuera de la "bolsa" son las muestras "out-of-bag", las que no se usan. Este parámetro indica si se usan muestras de bootstrap. Lo dejamos activado.

- **oob_score** (por defecto = False)

  Indica si se usarán muestras out-of-bag para estimar la precisión de generalización. En nuestro caso lo activamos, de esta manera usará también las muestras que no se usan en el procedimiento de agregación de bootstrap.

- **n_jobs** (por defecto = 1)

  El número de jobs o trabajos que se ejecutarán en paralelo tanto para fit (ajuste) como para predict (predicción), al ser 1 solo se ejecutará de manera secuencial, se podría mejorar el rendimiento aumentando el número de hilos, si la CPU del ordenador donde se ejecuta tiene varios núcleos se puede usar el valor -1 para que use como valor el número de núcleos que tiene la CPU y así se ejecutará cada trabajo en un núcleo distinto agilizando el proceso.

- **warm_start** (por defecto = False)

  Permite reutilizar el árbol creado previamente por la función fit (ajuste), como solo la llamaremos una vez no tendrá ningún efecto por lo que lo dejamos desactivado.

4.3.4. Ejemplos de uso

El proyecto está compuesto por tres scripts Python que se encargan de distintas tareas:

- **pcap_to_csv.py**
  Se encarga del preprocesado de los datos, lee las capturas ".pcapng" generadas por Wireshark y genera el fichero CSV con los datos extraídos.

- **reduce_capture.py**
  Lee el fichero CSV y genera varios CSV con distintos números de paquetes [100, 200, 500, 1000, 2000 y 5000].

- **main.py**
  El script principal, se encarga de leer los ficheros CSV, ejecutar los 3

algoritmos de Machine Learning sobre esos datos y generar los gráficos con los resultados.

Se pueden ejecutar por separado pasando los parámetros adecuados o usando el script **run.bat** (en Windows) o **run.sh** (en UNIX). El script contiene todas las instrucciones para ejecutar el proyecto por completo, por lo que el proceso llevará un largo tiempo aunque esto depende de las características del ordenador donde se ejecute. Es recomendable abrir el script y ejecutar las instrucciones por separado para poder ejecutar solo las que son de nuestro interés.

El resultado de la ejecución completa se representará en los gráficos que analizamos en este trabajo. Estas gráficas representan la precisión de cada algoritmo, la tasa de acierto/fallo.

5. Pruebas y/o resultados

Como se ha explicado en la sección anterior, se han probado 3 algoritmos de Machine Learning: **SVM**, **Decision Tree** y **Random Forest**.

La elección de estos algoritmos se debe a que, como hemos visto en el estudio previo [2. Análisis previo], estos algoritmos son los más usados o los que obtienen mejores resultados.

Para llevar a cabo las pruebas, se ha ejecutado el código que simplemente entrena los algoritmos con datos de entrenamiento y seguidamente comprueba con el resto del dataset si las clases predichas se corresponden con la etiqueta indicada en el dataset.

Tras realizar las pruebas se obtienen los resultados que analizamos a continuación.

5.1. Distribución de los datos

Antes de realizar las pruebas de los algoritmos, comprobamos la distribución de los datos según distintos pares de parámetros. Estas gráficas muestran visualmente como están distribuidos los paquetes cuando para un valor del parámetro en el eje X se tiene otro valor del parámetro en el eje Y. Además, los paquetes se colorean según la clase a la que pertenecen indicada por el campo etiqueta en el dataset.

Con 100 paquetes:

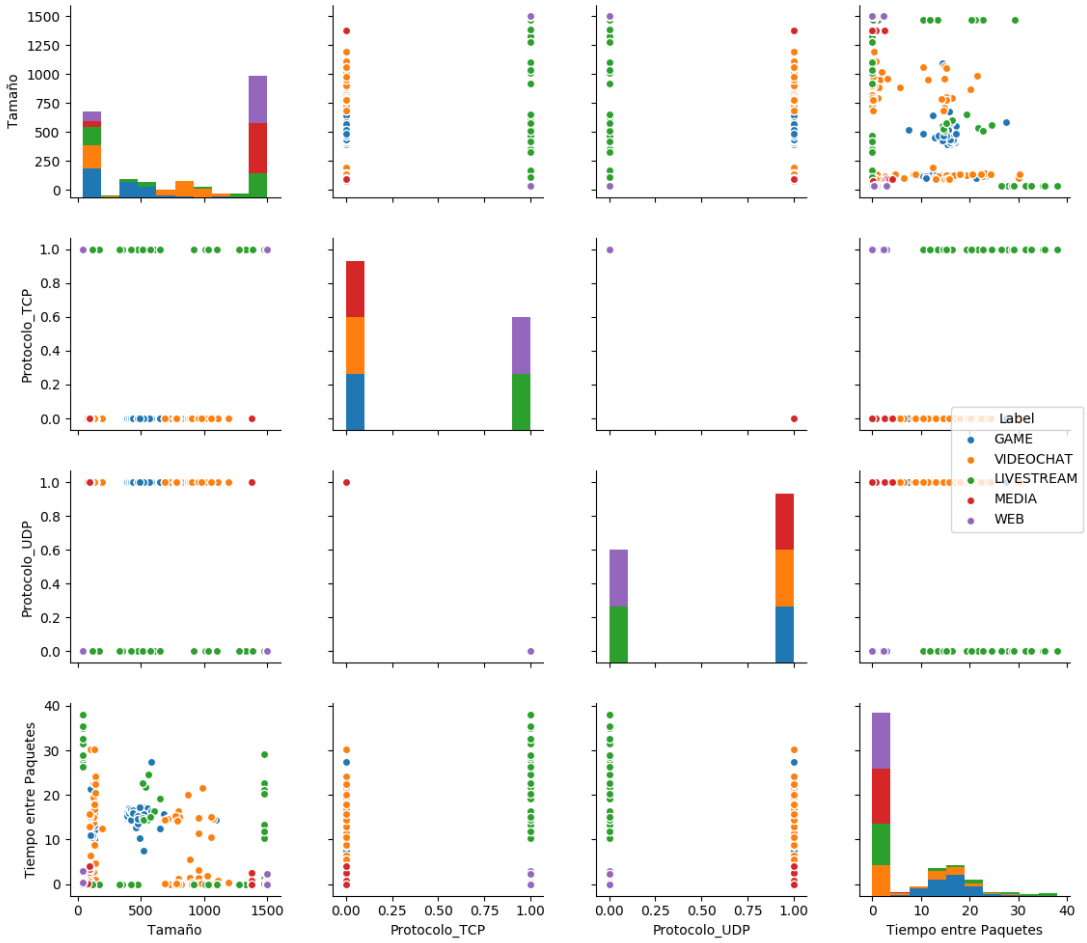

Figura 4: Distribución de los datos con 100 paquetes por clase.

Podemos comprobar que en la mayoría de los casos el tráfico LIVESTREAM y WEB se separan del resto, por ejemplo, en los campos Protocolo_TCP y Protocolo_UDP podemos observar que el tráfico GAME, VIDEOCHAT y MEDIA usa el protocolo UDP mientras que el tráfico LIVESTREAM y WEB usan el protocolo TCP por lo que en la gráfica aparecen visualmente divididos en dos grupos claramente diferenciados. En otros campos como el tamaño o el tiempo entre paquetes es más difícil ver una clara separación, sin embargo, se puede ver que el tiempo entre paquetes en la mayoría de paquetes es 0 probablemente debido a que cuando el paquete es el primero del flujo ponemos este campo a 0 por lo que si hay

muchos flujos y pocos paquetes pertenecientes a un mismo flujo, es decir, muchas comunicaciones pero de corta duración, la mayoría de los paquetes tendrán el campo tiempo entre paquetes con valor 0. Los únicos tipos de tráfico que muestran tener algunos flujos con más paquetes son GAME, VIDEOCHAT y LIVESTREAM lo cual tiene sentido ya que el tráfico que se transmite en un juego multijugador, en una video-llamada y al retransmitir un vídeo en directo enviará y/o recibirá muchos paquetes entre los mismos sockets, sin embargo, los paquetes del tráfico WEB y MEDIA parecen tener flujos más cortos, esto es normal en WEB ya que el tráfico simplemente envía una petición a un servidor y recibe la página como respuesta, no necesitan comunicarse más pero resulta un poco raro en MEDIA, quizás se deba a la muestra tan pequeña escogida (tan solo 100 paquetes) o a que la duración de los vídeos era demasiado corta para tener algún efecto sobre todo comparado con los otros tipos de tráfico como GAME y VIDEOCHAT que contienen una comunicación mucho más larga.

Con 200 paquetes:

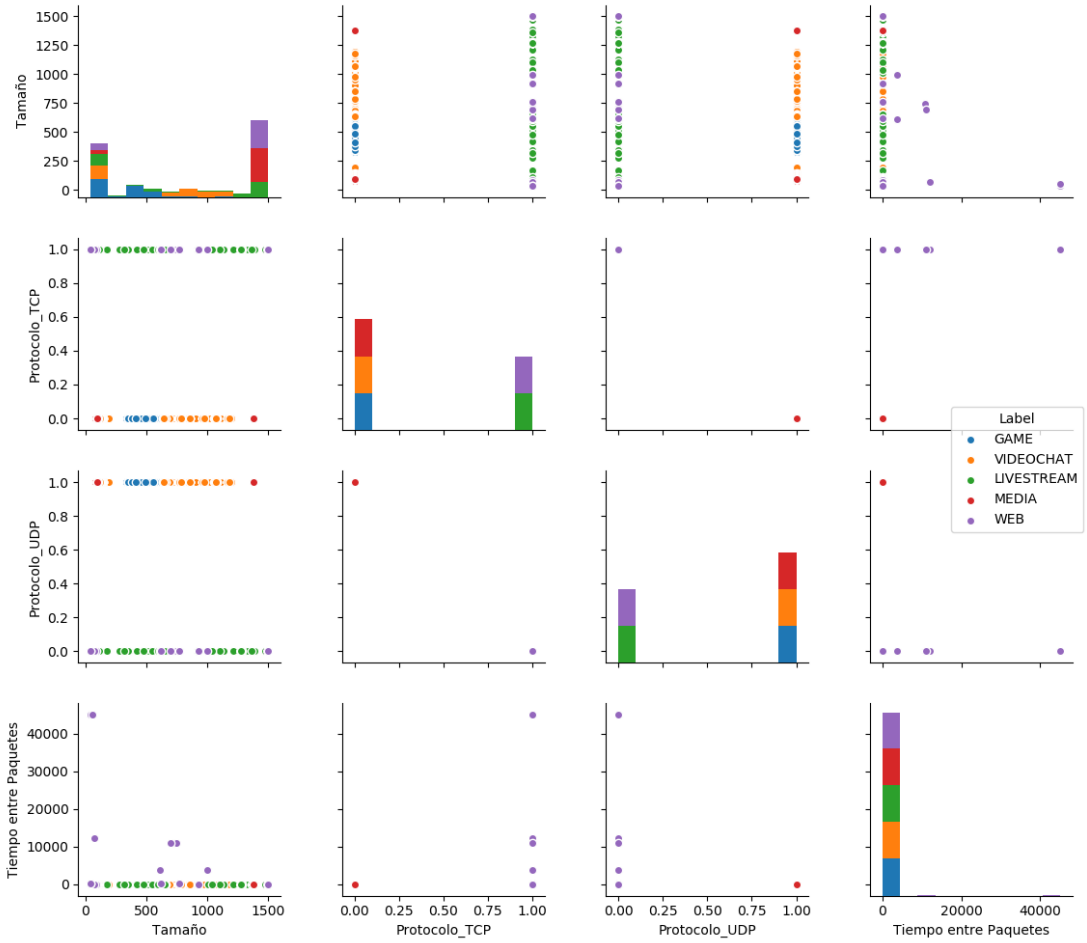

Figura 5: Distribución de los datos con 200 paquetes por clase.

Con una muestra de 200 paquetes por clase podemos observar una distribución muy parecida a la obtenida con 100 paquetes por clase, sin embargo, podemos ver una importante diferencia en el tiempo entre paquetes, ahora todos los paquetes tienden a 0, la muestra sigue siendo bastante pequeña pero un paquete con un tiempo entre paquetes muy grande hace que la gráfica muestre todo el peso en el valor 0, por esta razón parece que este campo no va a ser útil a la hora de clasificar el tráfico. Cabe destacar que los paquetes con tiempo superior son los del tráfico WEB, quizás se deba al lento tiempo de transmisión ya que al ser tráfico WEB no necesita responder

tan rápidamente como lo haría un vídeo en directo en el tráfico MEDIA y los paquetes que tarden demasiado se descartarían.

También podemos observar que el tamaño está bastante repartido entre las clases siendo las clases WEB y MEDIA las que tienen un tamaño más grande, esto quizás se deba a que las páginas web y los vídeos se estén transmitiendo en paquetes grandes mientras que otras aplicaciones como la del tráfico GAME que necesita responder en tiempo real prefiere enviar más paquetes de menor tamaño cada uno. Aun así, es importante mencionar que no hay una gran cantidad de paquetes con un tiempo entre paquetes tan alto por lo que es muy probable que sean casos aislados y no van a tener un gran efecto a la hora de clasificar.

Con 500 paquetes:

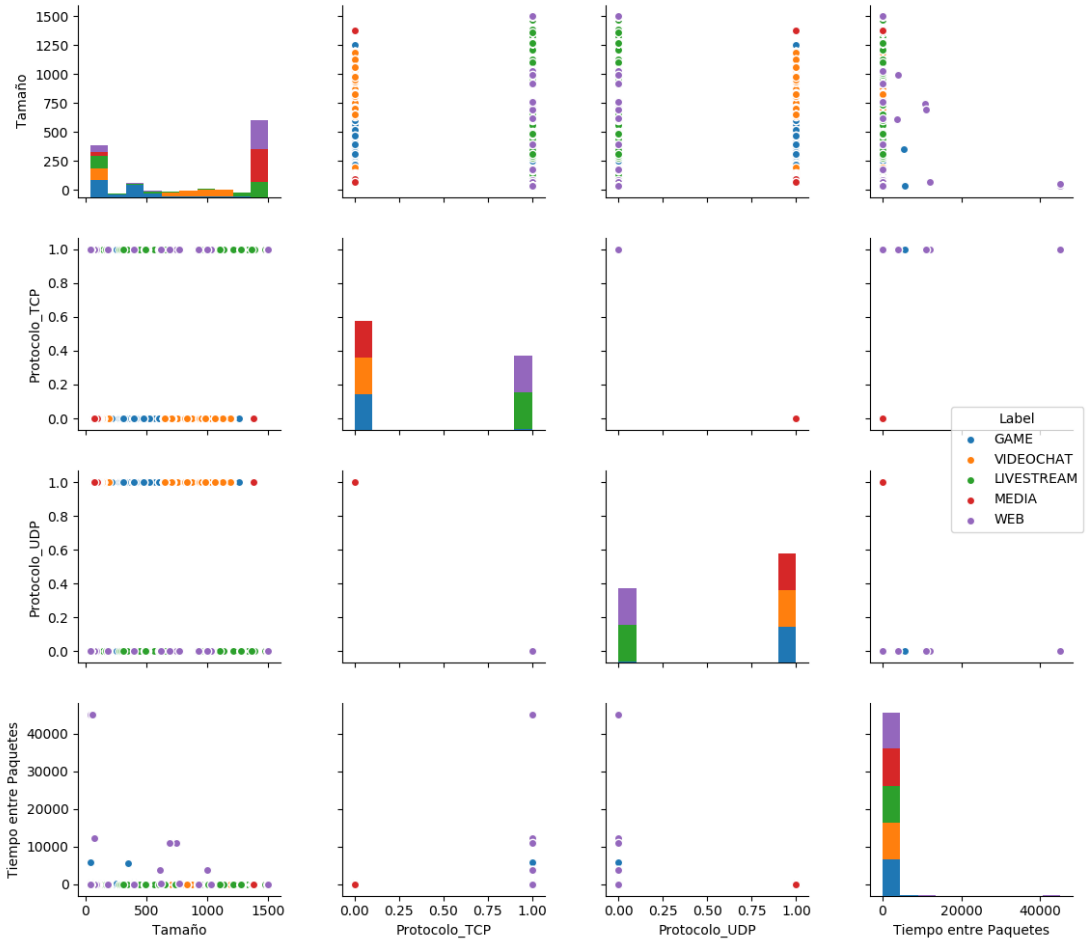

Figura 6: Distribución de los datos con 500 paquetes por clase.

La distribución con 500 paquetes por clase sigue siendo muy similar a la distribución con 200 paquetes y no hay cambios destacables.

Con 1000 paquetes:

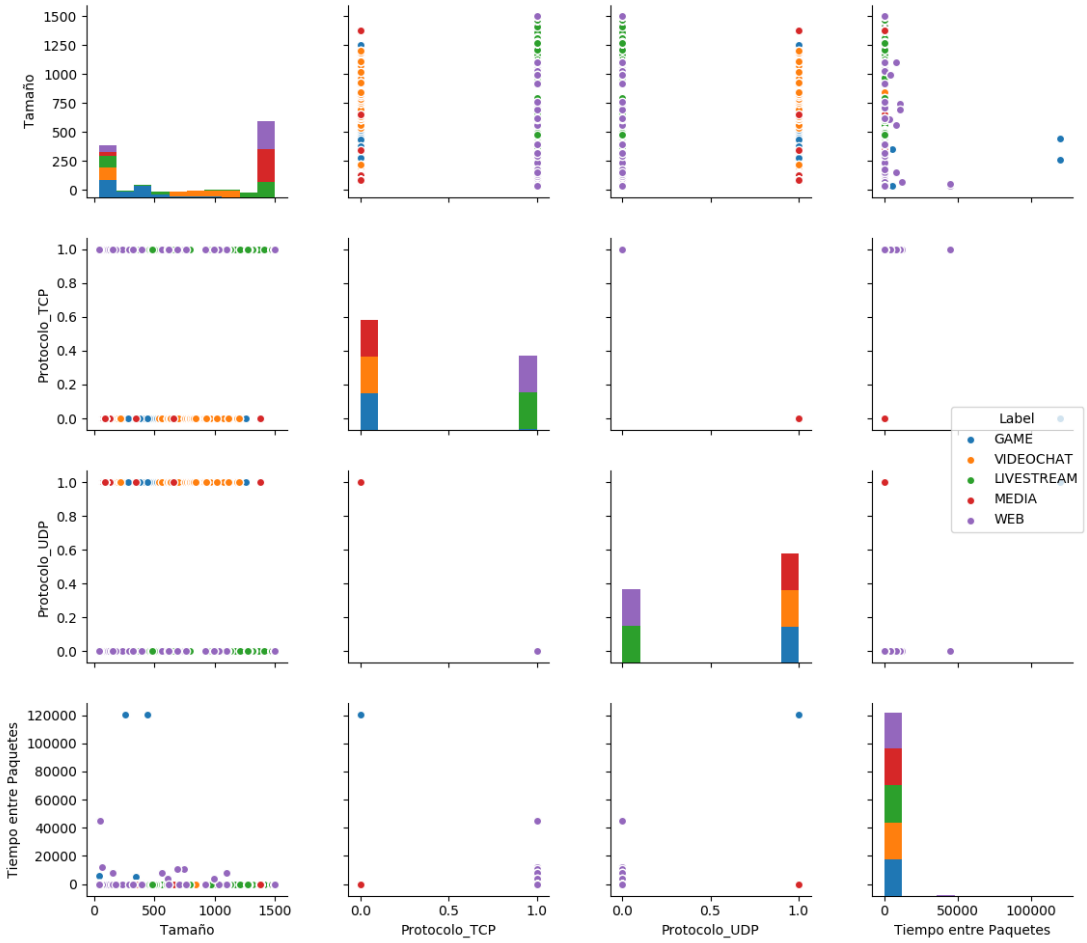

Figura 7: Distribución de los datos con 1000 paquetes por clase.

La distribución con 1000 paquetes por clase es muy similar a la distribución con 500 paquetes, sin embargo, podemos observar que ahora los paquetes con mayor tiempo entre paquetes pertenecen al tráfico de tipo GAME, de nuevo son casos muy concretos que no tienen relevancia.

Con 2000 paquetes:

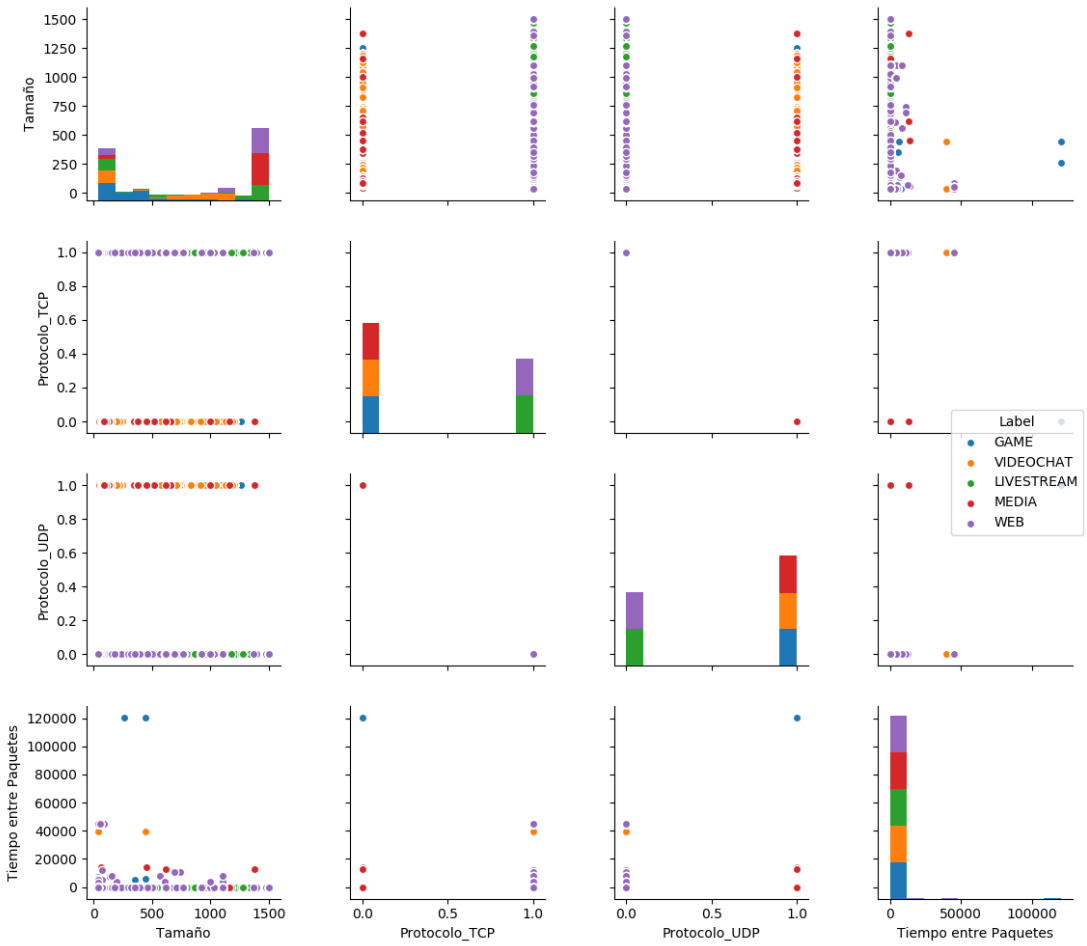

Figura 8: Distribución de los datos con 2000 paquetes por clase.

Con 2000 paquetes por clase obtenemos una distribución muy similar a la distribución con 1000 paquetes por clase. Podemos empezar a ver que a partir de 200 paquetes no ha habido grandes cambios y no parece que los vaya a haber aunque sigamos aumentando el número de paquetes.

Con 5000 paquetes:

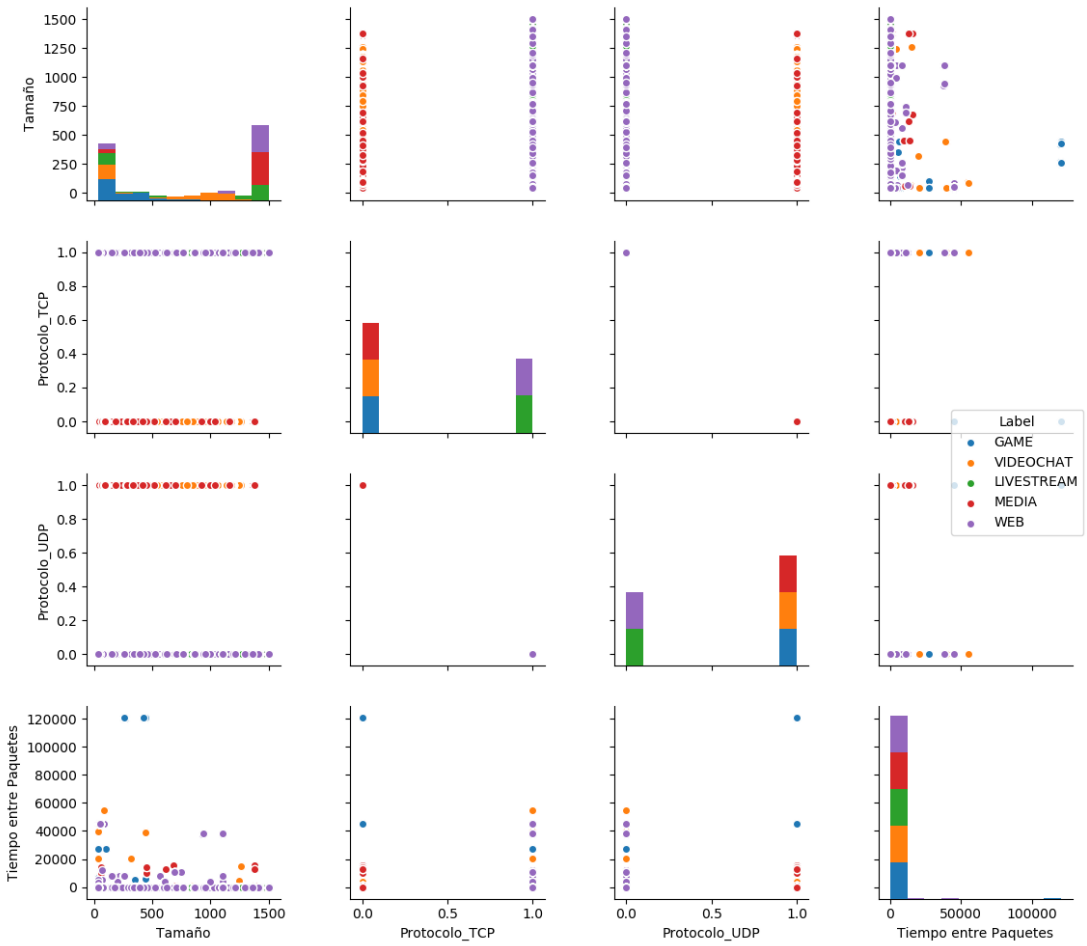

Figura 9: Distribución de los datos con 5000 paquetes por clase.

Finalmente, con el número máximo de paquetes por clase podemos observar que la distribución no ha cambiado mucho desde la distribución con 200 paquetes por clase. En general, los campos que parecen tener importancia son los que marcan el protocolo y quizás el tamaño, pero el tiempo entre paquetes no parece ser relevante.

Si ignoramos los casos aislados, podemos comprobar que la distribución no ha cambiado mucho incluso desde los 100 paquetes por clase. El tráfico GAME, VIDEOCHAT y MEDIA usa el protocolo UDP y el tráfico WEB y LIVESTREAM usa el protocolo TCP, esa división es muy clara y ha permanecido igual para todas las distribuciones. El tamaño de los paquetes parece ser más grande en el tráfico

WEB y en algún que otro paquete del tráfico MEDIA pero es muy inferior en el tráfico de tipo VIDEOCHAT y en el de tipo GAME, esto puede deberse como se mencionó anteriormente a que el tráfico WEB envía todo el contenido en pocos paquetes y no le importa tardar más en llegar, mientras que en tráfico en tiempo real como GAME prefieren enviar paquetes más pequeños en más cantidades por lo que el flujo permanece abierto más tiempo pero los mensajes tardan menos en llegar.

5.2. Resultados para SVM

A continuación vemos los resultados obtenidos con el algoritmo SVM. Para ello usamos un tipo de gráfica que representa la matriz de confusión. Esta gráfica indica para la clase del eje Y (la clase correcta según la etiqueta del dataset) en qué clase la ha clasificado el algoritmo indicado por el eje X. Es decir, si en el eje Y tenemos la clase WEB, podemos ver cuántos paquetes WEB ha clasificado como la clase indicada en el eje X con un número de 0 a 1 siendo 0 un 0% y 1 un 100%. También se indican visualmente con colores el porcentaje, usando un color oscuro o morado para los valores más bajos y un color más claro o naranja para los valores más altos. Si en el eje Y tenemos la clase WEB y en el eje X también tenemos la clase WEB, para que el algoritmo haya clasificado correctamente todos los paquetes de tipo WEB debe indicar un 1 en esta posición y un 0 para las otras clases del eje X, si no es así se puede ver en qué otras clases ha clasificado incorrectamente estos paquetes.

Con 100 paquetes:

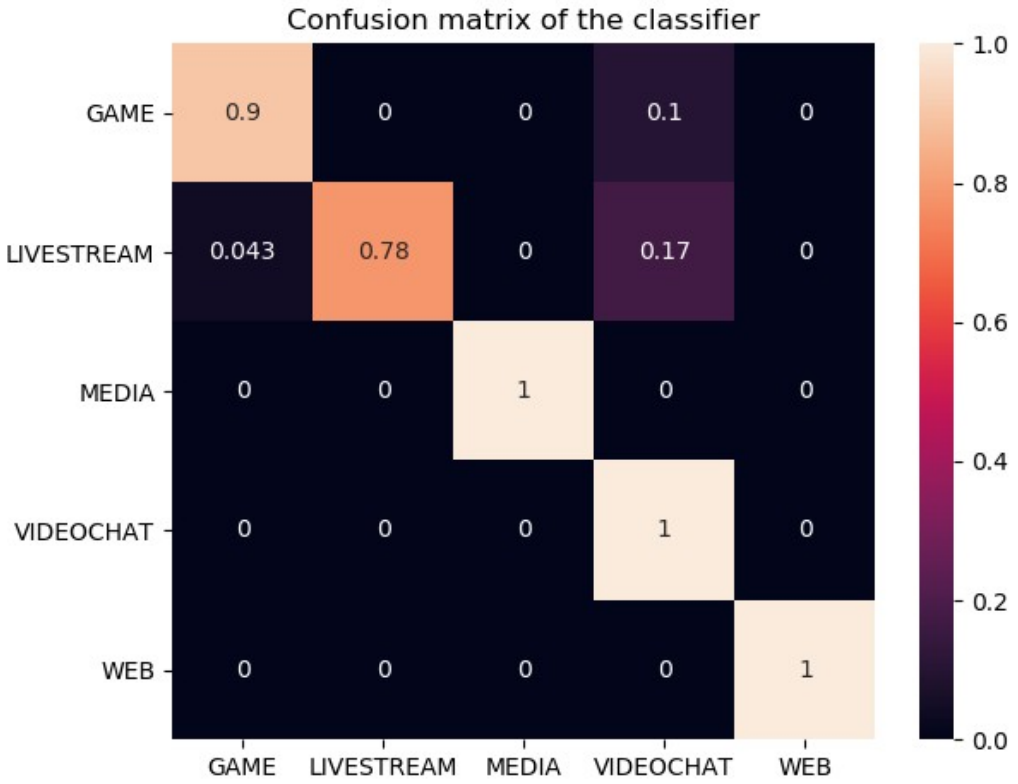

Figura 10: Matriz de confusión para SVM con 100 paquetes por clase.

Podemos observar que el algoritmo falla al clasificar algunos paquetes, el 100% de los paquetes de tipo MEDIA, VIDEOCHAT y WEB los clasifica correctamente, pero de los de tipo GAME solo clasifica bien el 90%, el 10% restante lo clasifica incorrectamente como VIDEOCHAT. Un fallo del 10% no parece tan grave, sin embargo, de los paquetes de tipo LIVESTREAM solo el 78% son clasificados correctamente y esto ya es un margen de error bastante importante, el 17% los clasifica incorrectamente como VIDEOCHAT y un 4.3% como GAME.

Con 200 paquetes:

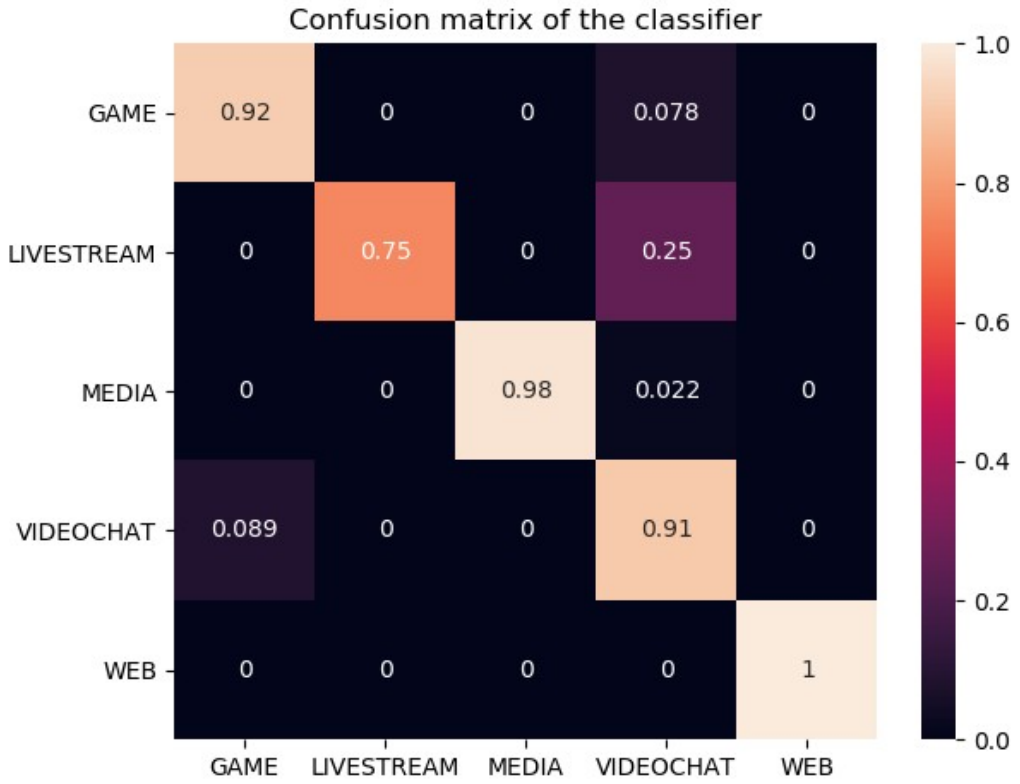

Figura 11: Matriz de confusión para SVM con 200 paquetes por clase.

Con 200 paquetes sigue teniendo bastantes fallos, en este caso solo en WEB clasifica correctamente el 100% de paquetes, aunque GAME, MEDIA y VIDEOCHAT tienen un margen de acierto bastante aceptable (superior al 90%). Igual que con 100 paquetes, LIVESTREAM sigue teniendo demasiado fallo (solo acierta el 75% de paquetes) y podemos ver como los paquetes que clasifica incorrectos en todas las clases los suele asociar con VIDEOCHAT.

Con 500 paquetes:

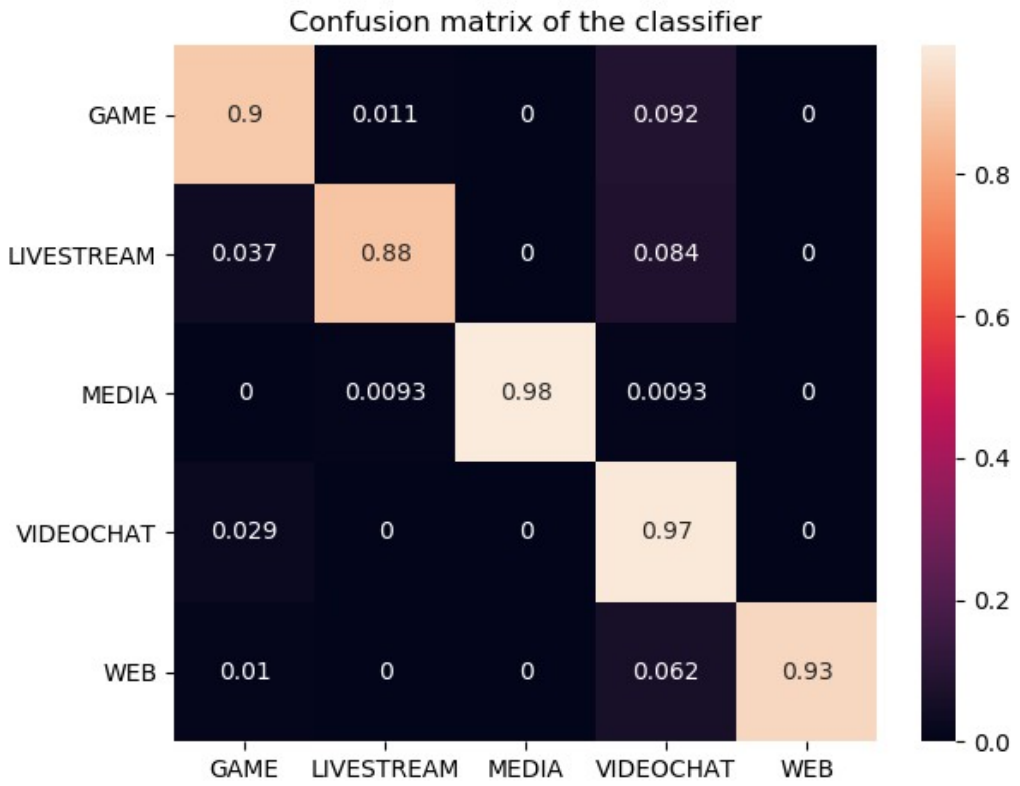

Figura 12: Matriz de confusión para SVM con 500 paquetes por clase.

Con 500 paquetes ya empieza a dar mejores resultados, aunque ninguno perfecto. Todas las clases excepto LIVESTREAM tienen una buena tasa de acierto (más del 90%) con pequeños fallos despreciables. LIVESTREAM que en los dos casos anteriores tenía una tasa de fallo elevado ha mejorado considerablemente aunque sigue por debajo del 90%. Y los paquetes que clasifica incorrectamente los sigue asociando a VIDEOCHAT.

Con 1000 paquetes:

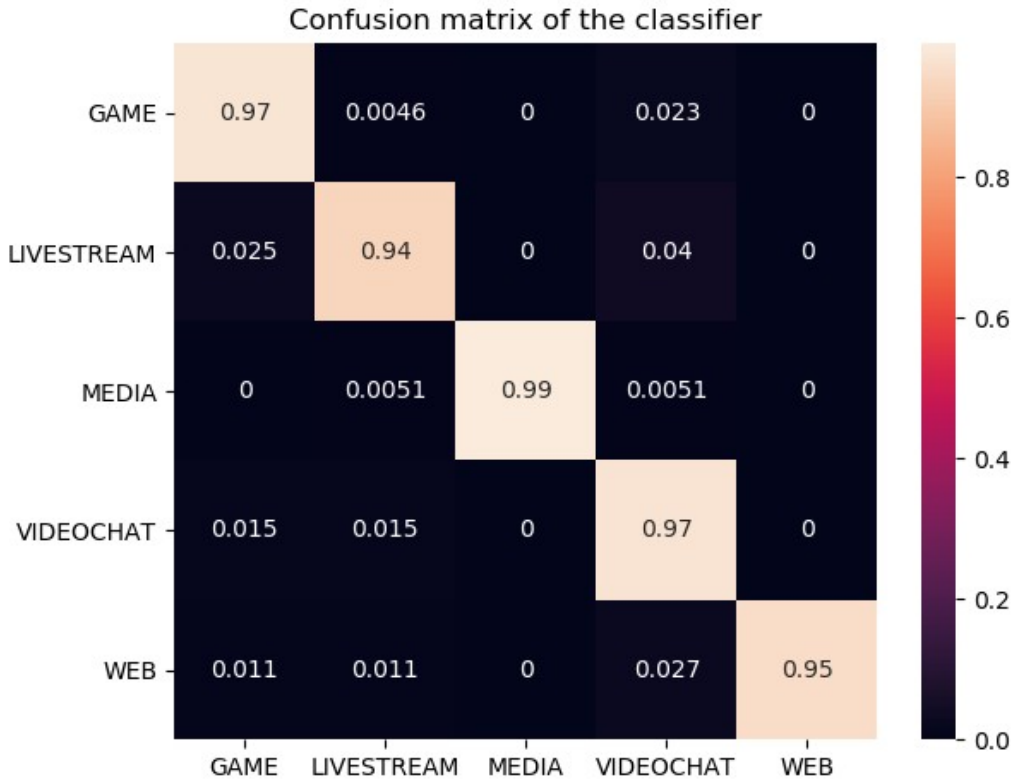

Figura 13: Matriz de confusión para SVM con 1000 paquetes por clase.

Con 1000 paquetes ya hay una gran tasa de acierto en todas las clases (todas superan el 90%) y los fallos son despreciables. La clase que sigue teniendo más fallos es LIVESTREAM aunque tiene un 94% de acierto y en caso de fallo, suele clasificar el 4% como VIDEOCHAT que sigue siendo la clase más usada en caso de fallo.

Con 2000 paquetes:

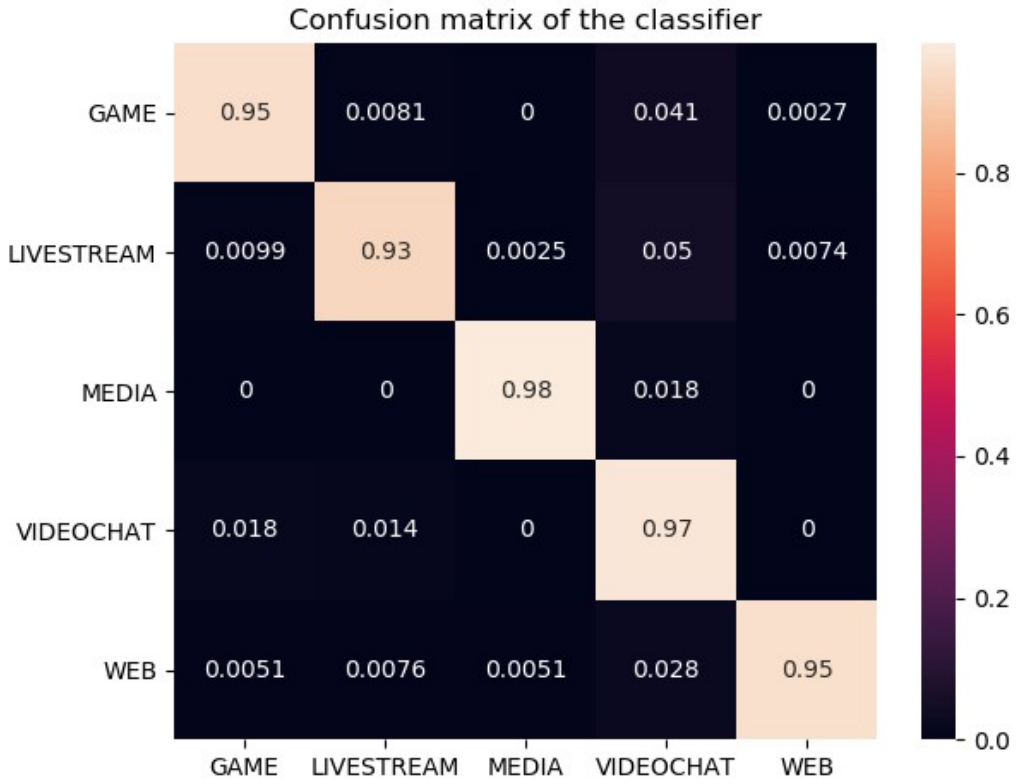

Figura 14: Matriz de confusión para SVM con 2000 paquetes por clase.

Los resultados con 2000 paquetes son equivalentes a los resultados con 1000. Toda las clases tienen una tasa de acierto superior al 90%.

Con 5000 paquetes:

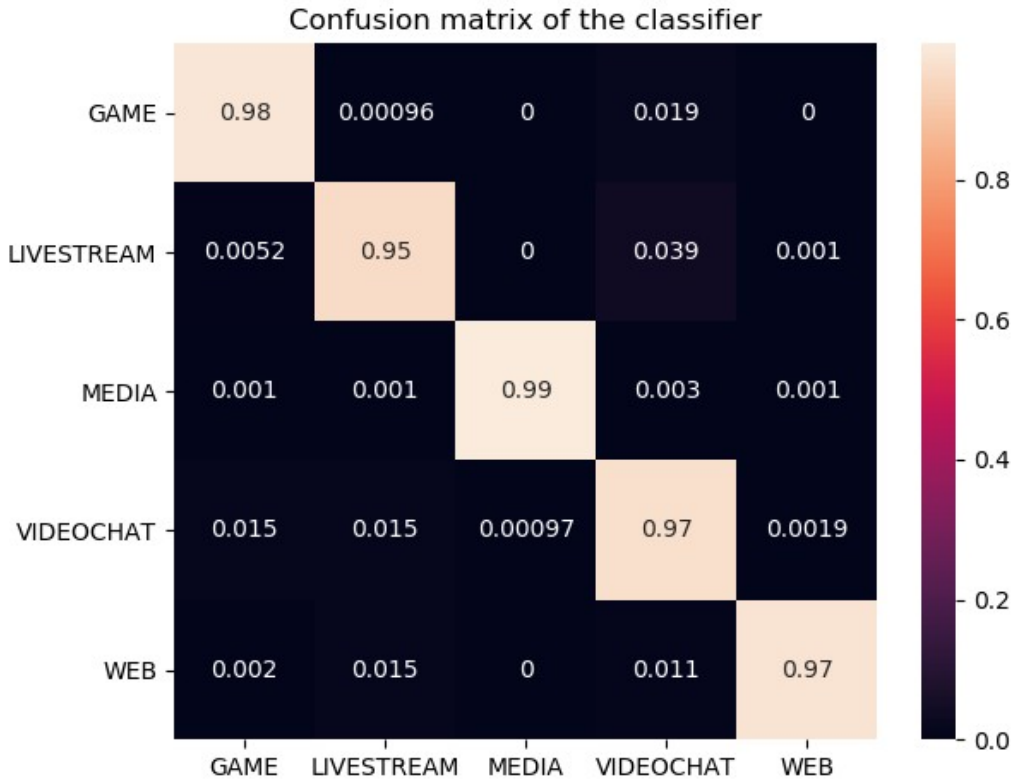

Figura 15: Matriz de confusión para SVM con 5000 paquetes por clase.

Con 5000 paquetes volvemos a tener un resultado equivalente a con 1000 y 2000 paquetes, aunque usemos más paquetes no mejora la situación, sin embargo, la tasa de acierto ya es muy elevada y, aunque no es perfecta, alcanza en todos los casos el 95%.

5.3. Resultados para Decision Tree

A continuación vemos los resultados obtenidos con el algoritmo Decision Tree. Para ello volvemos a usar el tipo de gráfica que usamos con el algoritmo SVM que representa la matriz de confusión. Esta gráfica indica para la clase del eje Y (la clase correcta según la etiqueta del dataset) en qué clase la ha clasificado el algoritmo indicado por el eje X. Es decir, si en el eje Y tenemos la clase WEB, podemos ver cuántos paquetes WEB ha clasificado como la clase indicada en el eje X con un número de 0 a 1 siendo 0 un 0% y 1 un 100%. También se indican visualmente con colores el porcentaje, usando un color oscuro o morado para los valores más bajos y un color más claro o naranja para los valores más altos. Si en el eje Y tenemos la clase WEB y en el eje X también tenemos la clase WEB, para que el algoritmo haya clasificado correctamente todos los paquetes de tipo WEB debe indicar un 1 en esta posición y un 0 para las otras clases del eje X, si no es así se puede ver en qué otras clases ha clasificado incorrectamente estos paquetes.

Además, para este algoritmo podemos ver qué árbol se ha generado tras entrenarlo, esto se representa como un grafo binario en el que podemos ver para ciertos parámetros por qué camino avanza hasta concluir en una clase resultado.

Con 100 paquetes:

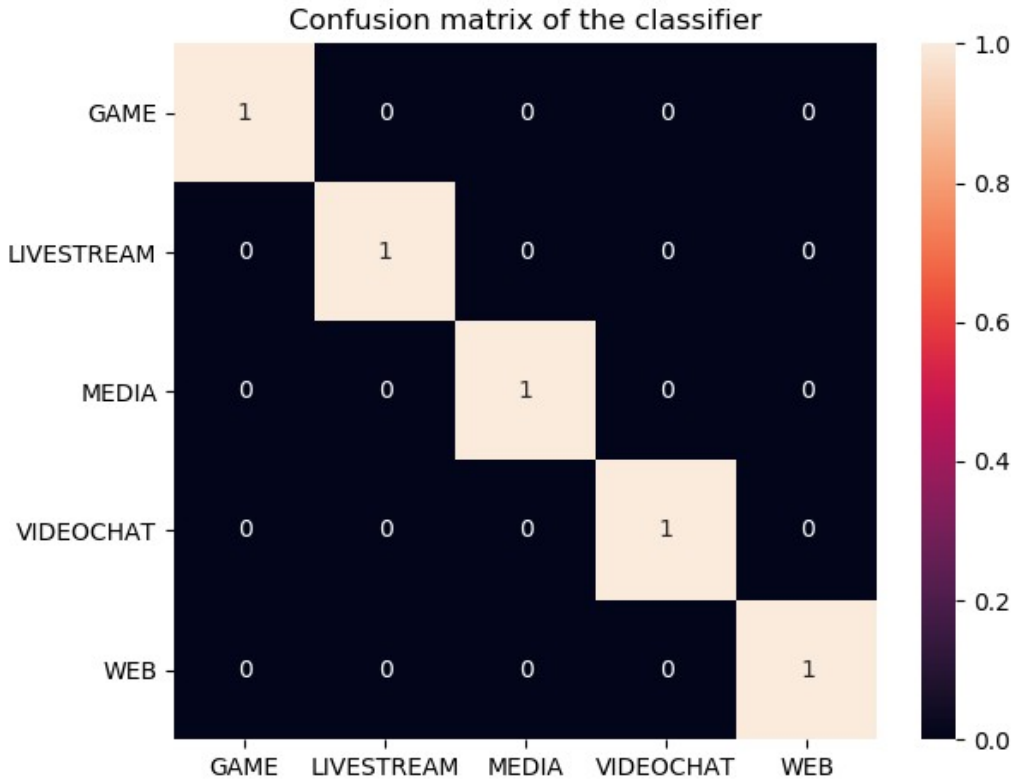

Figura 16: Matriz de confusión para Decision Tree con 100 paquetes por clase.

Con 100 paquetes los resultados son perfectos, una tasa de acierto del 100% en todas las clases. Esto quizás se deba a sobreajuste, la muestra al ser de tan solo 100 paquetes por clase parece demasiado pequeña.

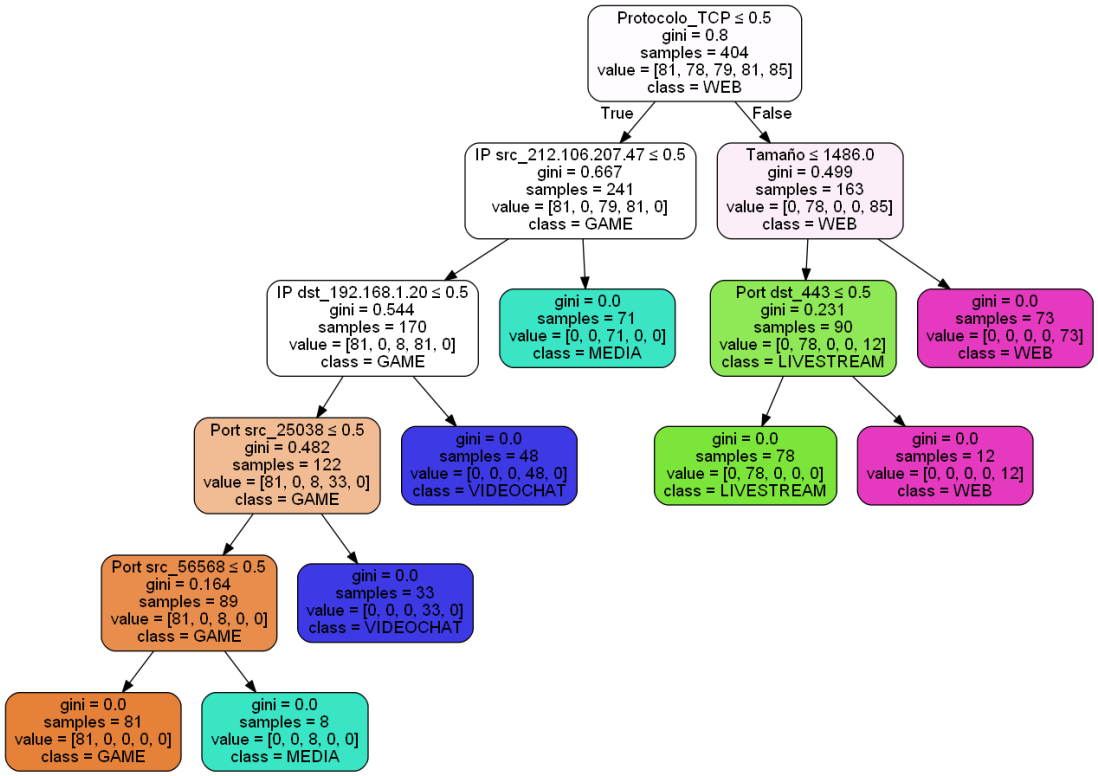

Figura 17: Representación de Decision Tree entrenado con 100 paquetes por clase.

Podemos observar que el árbol generado es bastante pequeño debido a que la muestra elegida es muy pequeña. Para predecir a qué clase pertenece un paquete empieza por la hoja superior y comprueba si las comparaciones son verdaderas o falsas, la primera comparación es Protocolo_TCP <= 0.5 que simplemente indica que si es verdadero el paquete usa el protocolo UDP y si es falso usa el protocolo TCP, esto se debe a que el campo Protocolo_TCP solo tiene como valores 0 (en el caso de que NO use el protocolo TCP) o 1 (en el caso de que SÍ lo use), funciona de la misma manera que el campo Protocolo_UDP y estos dos campos siempre tendrán el valor inverso entre sí, es decir, si un paquete usa el protocolo TCP el valor del campo Protocolo_TCP será 1 y el valor del campo Protocolo_UDP será 0. Por esta razón no tiene sentido usar los dos campos y se puede observar en el árbol que le basta con usar uno de ellos. Más adelante comprueba el tamaño y las direcciones IP hasta llegar a la última hoja y poder dar un resultado.

Con 200 paquetes:

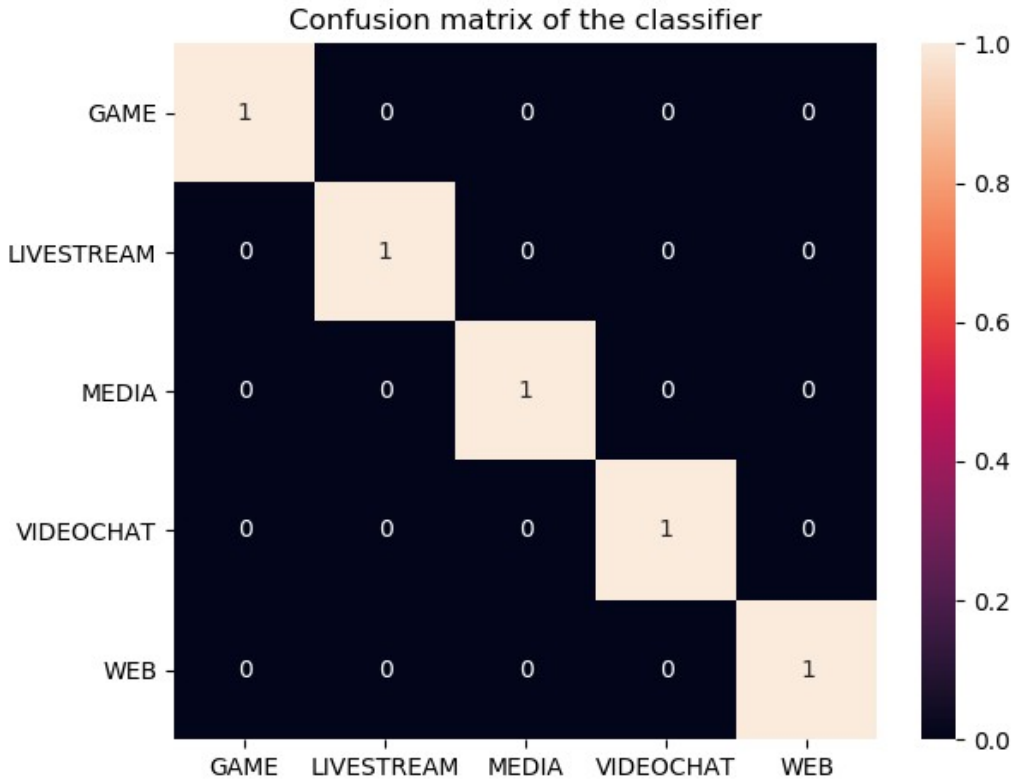

Figura 18: Matriz de confusión para Decision Tree con 200 paquetes por clase.

Con 200 también obtenemos una tasa de acierto del 100%. Sigue siendo un resultado muy perfecto por lo que quizás la muestra sea demasiado pequeña.

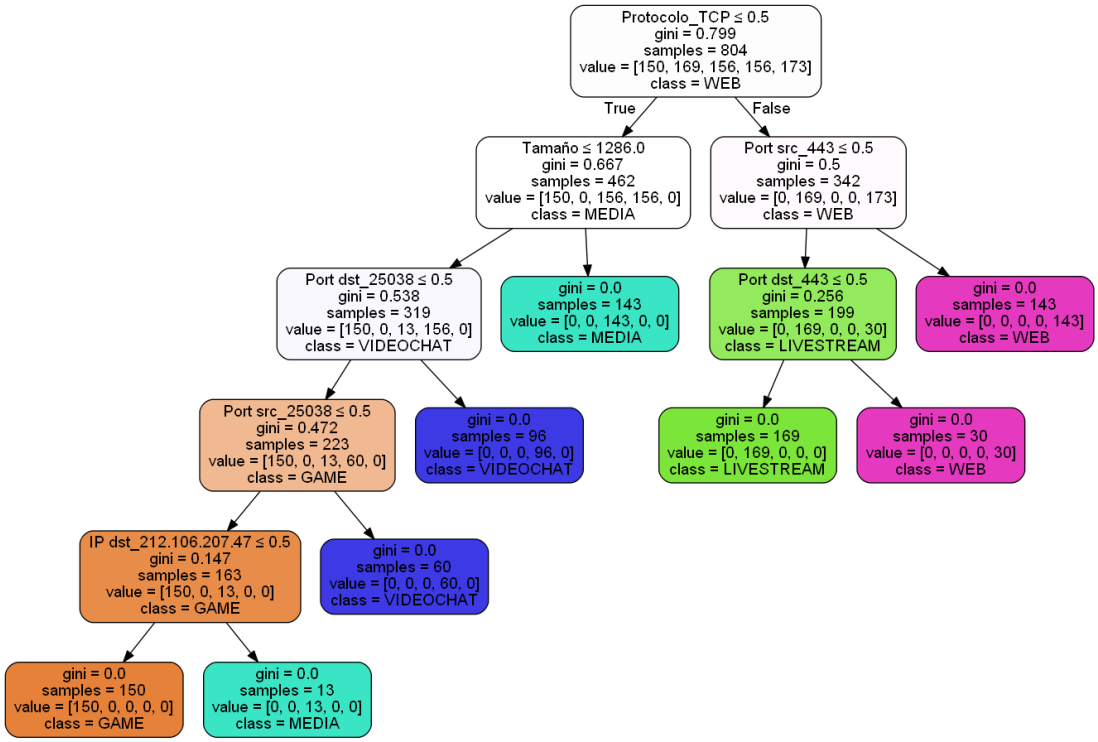

Figura 19: Representación de Decision Tree entrenado con 200 paquetes por clase.

Podemos comprobar que el árbol es más grande al tener una muestra más grande, al haber más paquetes se necesita una lógica más compleja para poder obtener el resultado correcto. Sin embargo, el árbol sigue siendo bastante pequeño. De la misma manera que en el caso anterior empieza comprobando si el protocolo es TCP o UDP, acto seguido comprueba el tamaño y el puerto y continúa hasta llegar a la última hoja.

Con 500 paquetes:

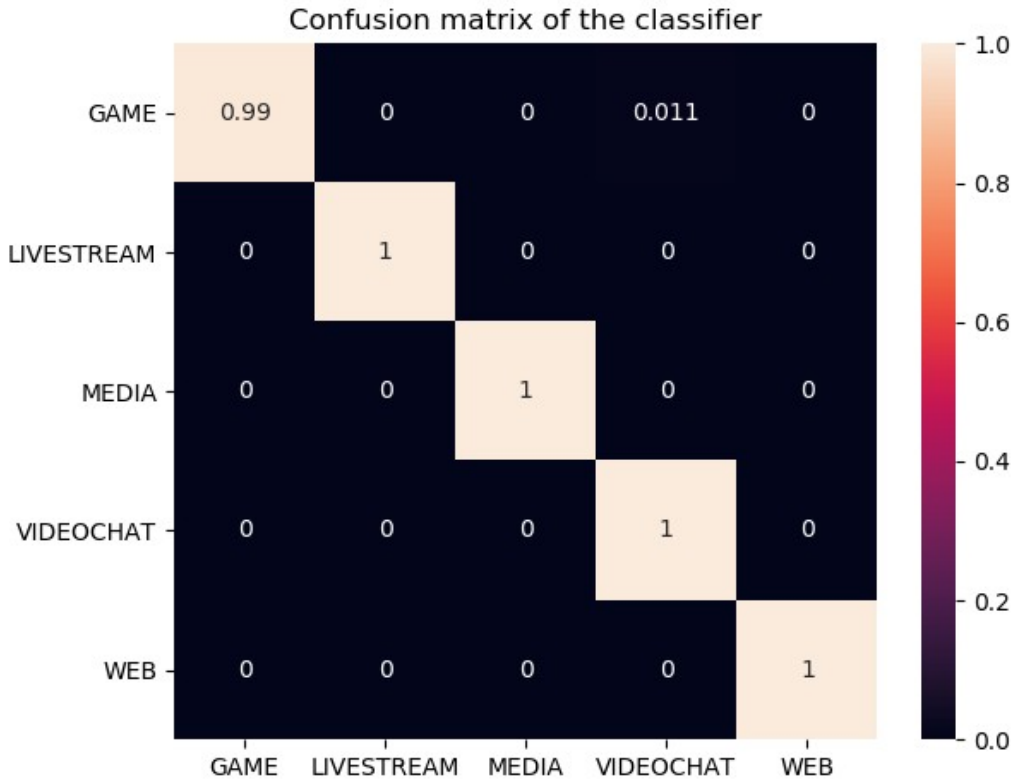

Figura 20: Matriz de confusión para Decision Tree con 500 paquetes por clase.

Con 500 paquetes obtenemos una tasa de acierto que es prácticamente del 100% aunque en la clase GAME tenemos un error del 1% que considero despreciable. Sigue siendo un resultado bastante perfecto.

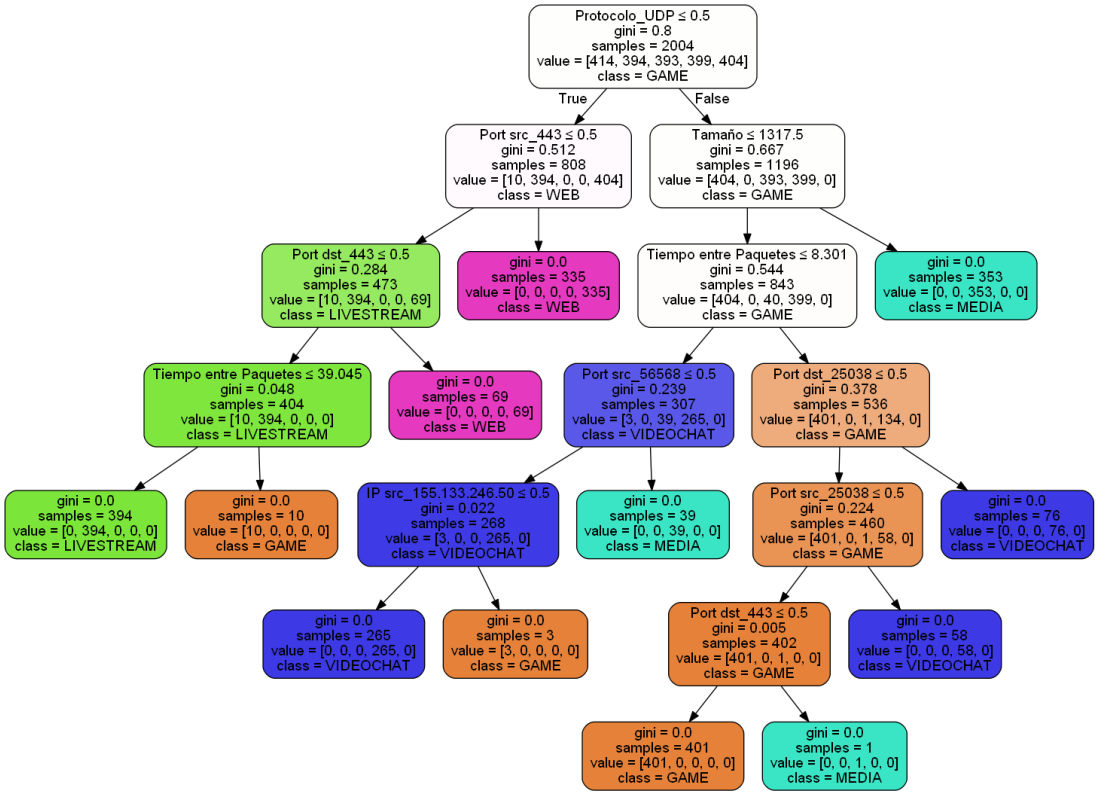

Figura 21: Representación de Decision Tree entrenado con 500 paquetes por clase.

Podemos observar que el árbol ahora es bastante más grande, pero sigue siendo pequeño debido a que la muestra sigue siendo demasiado pequeña y el tráfico de cada clase difiere lo suficiente entre sí como para no producir errores.

Con 1000 paquetes:

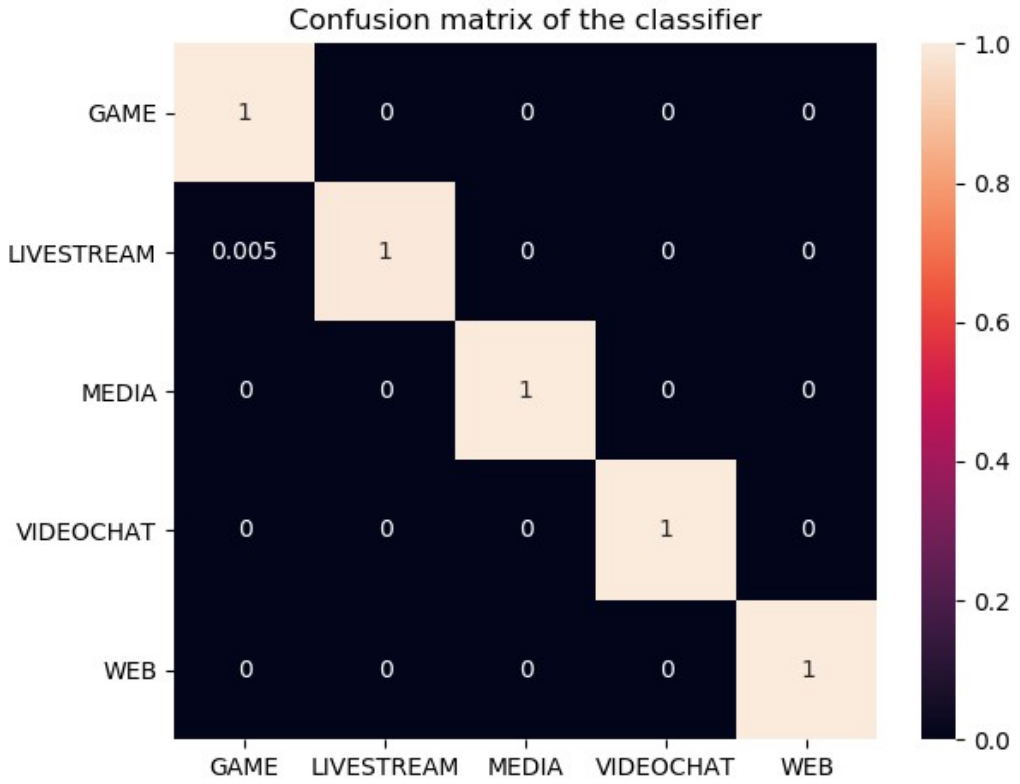

Figura 22: Matriz de confusión para Decision Tree con 1000 paquetes por clase.

Con 1000 paquetes seguimos teniendo una tasa de acierto prácticamente perfecta, aunque en LIVESTREAM hay algún paquete que clasifica incorrectamente como GAME (en el caso anterior también tenía algún fallo con la clase GAME). El fallo es despreciable ya que es del 0.5% y por redondeo desaparece. Esto quiere decir que en todas las pruebas llevamos obteniendo resultados perfectos, esto resulta bastante raro y es muy probable que se deba a que la muestra es demasiado pequeña o a que las clases están lo suficientemente diferenciadas como para que nunca pueda dar error.

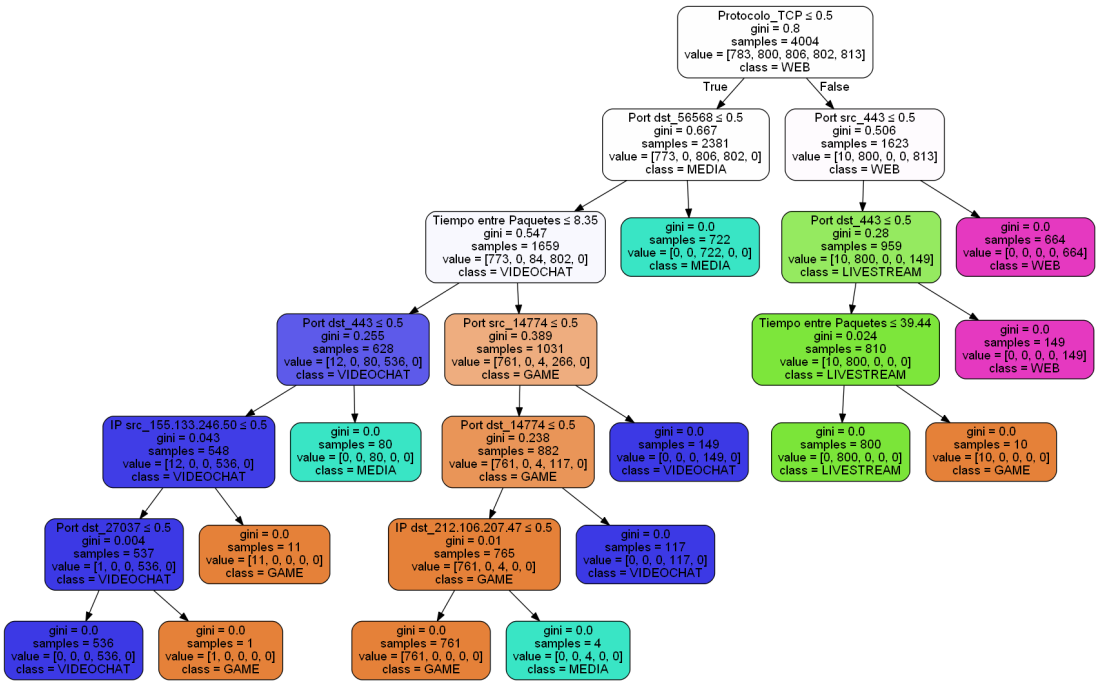

Figura 23: Representación de Decision Tree entrenado con 1000 paquetes por clase.

El árbol vuelve a aumentar al tener una muestra más grande, pero sigue siendo demasiado sencillo debido a que la muestra es pequeña. Con datos realmente complejos no debería ser posible tener un árbol tan sencillo con tan pocas comprobaciones para poder dar resultados tan perfectos.

Con 2000 paquetes:

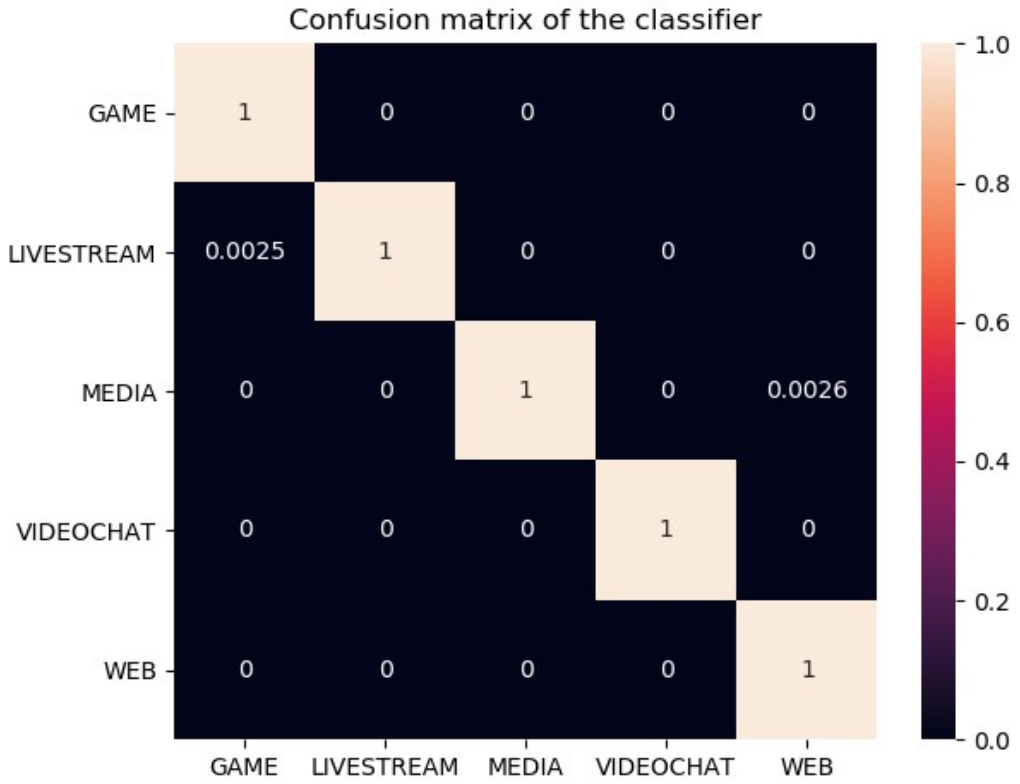

Figura 24: Matriz de confusión para Decision Tree con 2000 paquetes por clase.

Con 2000 paquetes seguimos teniendo un acierto perfecto, de la misma manera al caso anterior podemos apreciar algunos fallos despreciables.

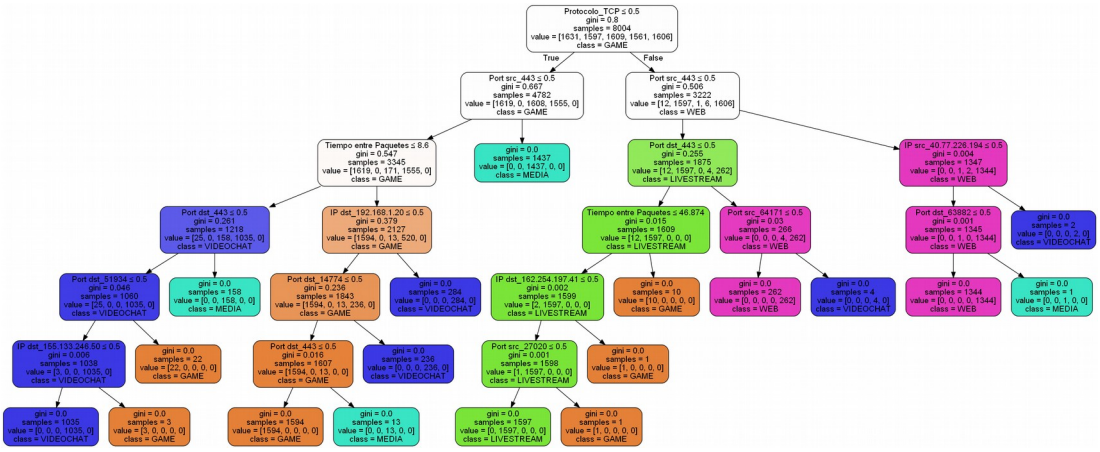

Figura 25: Representación de Decision Tree entrenado con 2000 paquetes por clase.

En la representación del árbol podemos ver como ya empieza a coger complejidad, sin embargo resulta demasiado simple y los resultados siguen saliendo demasiado perfectos.

Con 5000 paquetes:

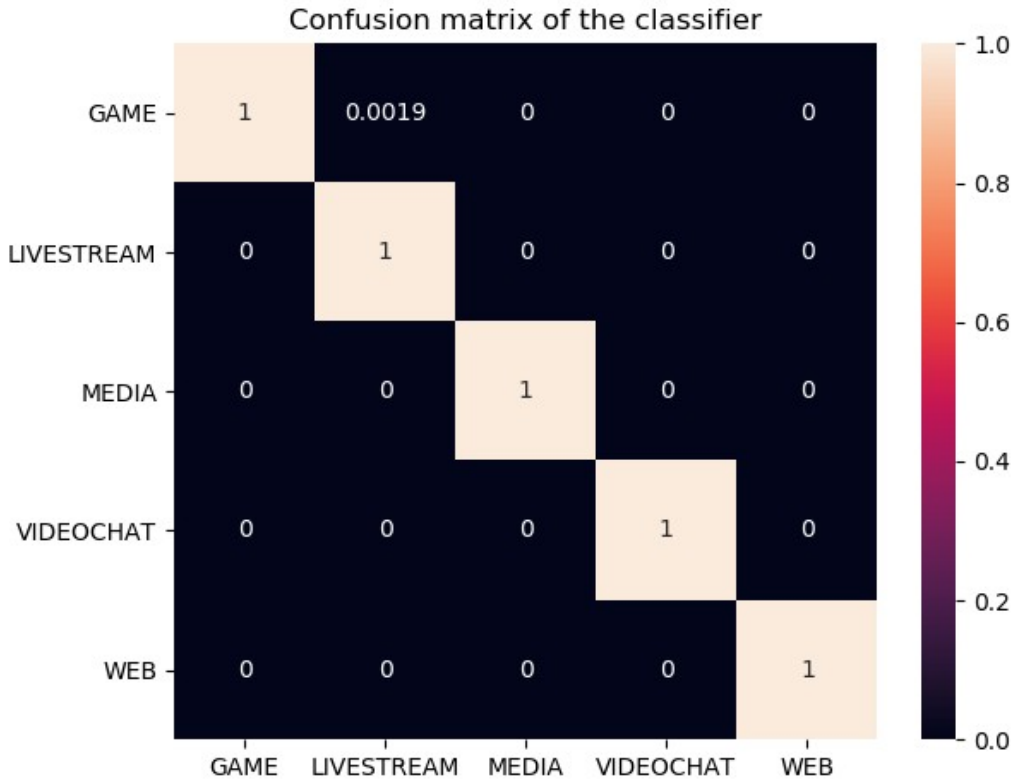

Figura 26: Matriz de confusión para Decision Tree con 5000 paquetes por clase.

Finalmente, con 5000 paquetes tenemos una tasa de acierto del 100% con fallos despreciables. Desde la primera prueba con 100 paquetes hemos tenido una tasa de acierto casi perfecta en todas las clases (tan cerca que por redondeo se acaba quedando en 100%, siempre bastante mayor de 99%). Con una tasa de fallo tan pequeña, se pueden despreciar los errores por lo que con solo 100 paquetes ya nos da resultados muy buenos y no parece que necesitemos grandes cantidades de datos para entrenar a la máquina. Sin embargo, es muy probable que estos resultados se deban a una muestra demasiado pequeña. Hay que tener en cuenta que el tráfico utilizado está muy bien diferenciado, cada clase usa tan solo una aplicación y son aplicaciones muy distintas entre sí por lo que en tráfico real los resultados no serán tan buenos.

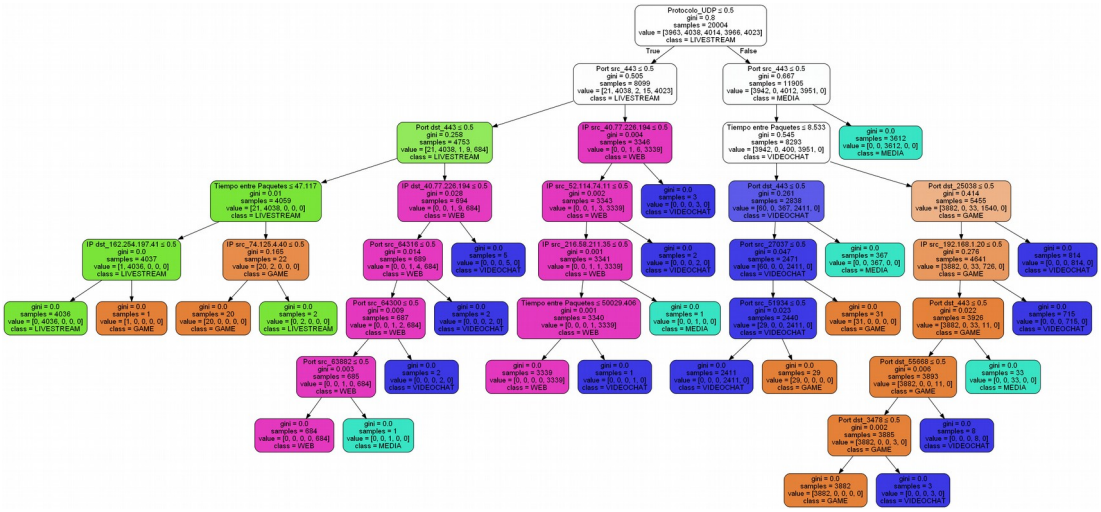

Figura 27: Representación de Decision Tree entrenado con 5000 paquetes por clase.

Finalmente, podemos observar como ha crecido el árbol para poder acertar la predicción para una muestra de 5000 paquetes. Sigue pareciendo bastante simple pero podemos observar como se complican las comprobaciones. Con tráfico real, resulta dudoso que se puedan obtener buenos resultados con un árbol tan adaptado al tráfico concreto con el que lo hemos entrenado.

5.4. Resultados para Random Forest

A continuación vemos los resultados obtenidos con el algoritmo Random Forest. Como en los algoritmos anteriores volvemos a usar el tipo de gráfica que representa la matriz de confusión. Esta gráfica indica para la clase del eje Y (la clase correcta según la etiqueta del dataset) en qué clase la ha clasificado el algoritmo indicado por el eje X. Es decir, si en el eje Y tenemos la clase WEB, podemos ver cuántos paquetes WEB ha clasificado como la clase indicada en el eje X con un número de 0 a 1 siendo 0 un 0% y 1 un 100%. También se indican visualmente con colores el porcentaje, usando un color oscuro o morado para los valores más bajos y un color más claro o naranja para los valores más altos. Si en el eje Y tenemos la clase WEB y en el eje X también tenemos la clase WEB, para que el algoritmo haya clasificado correctamente todos los paquetes de tipo WEB debe indicar un 1 en esta posición y

un 0 para las otras clases del eje X, si no es así se puede ver en qué otras clases ha clasificado incorrectamente estos paquetes.

Como se explicó en el apartado 4.3.3. Random Forest, este algoritmo usa varios árboles (creando así un bosque) y usando la media de los valores obtenidos como resultado, por esta razón no podemos obtener la representación de un árbol como se hizo para el algoritmo Decision Tree, ya que en este caso no tenemos un solo árbol sino muchos de ellos (en este estudio hemos usado 100 árboles).

Con 100 paquetes:

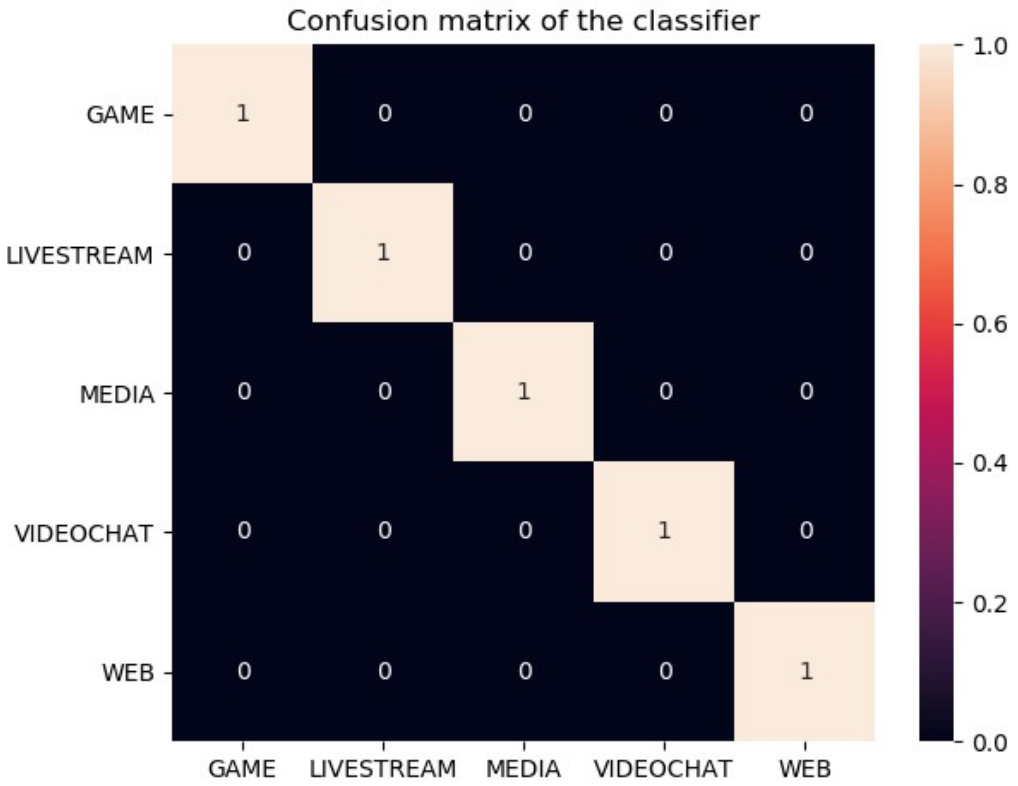

Figura 28: Matriz de confusión para Random Forest con 100 paquetes por clase.

Con 100 paquetes, el resultado es perfecto, la tasa de acierto es 100% para todas las clases y, por lo tanto, los fallos son del 0%. De nuevo obtenemos resultados perfectos como en Decision Tree, aunque tiene más sentido obtener estos resultados en Random Forest debido a que usa varios árboles y coge la media como resultado,

sigue siendo demasiado perfecto y es probable que se deba a la muestra tan pequeña elegida.

Con 200 paquetes:

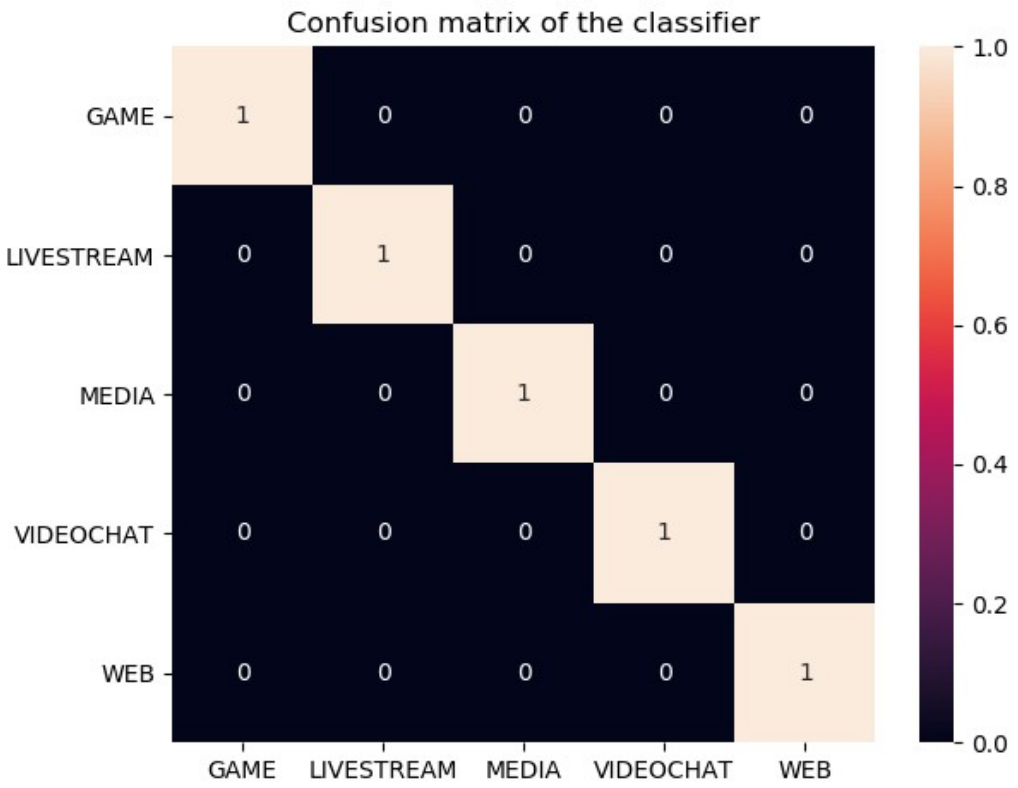

Figura 29: Matriz de confusión para Random Forest con 200 paquetes por clase.

Con 200 paquetes tenemos el mismo resultado, una tasa de acierto del 100%.

Con 500 paquetes:

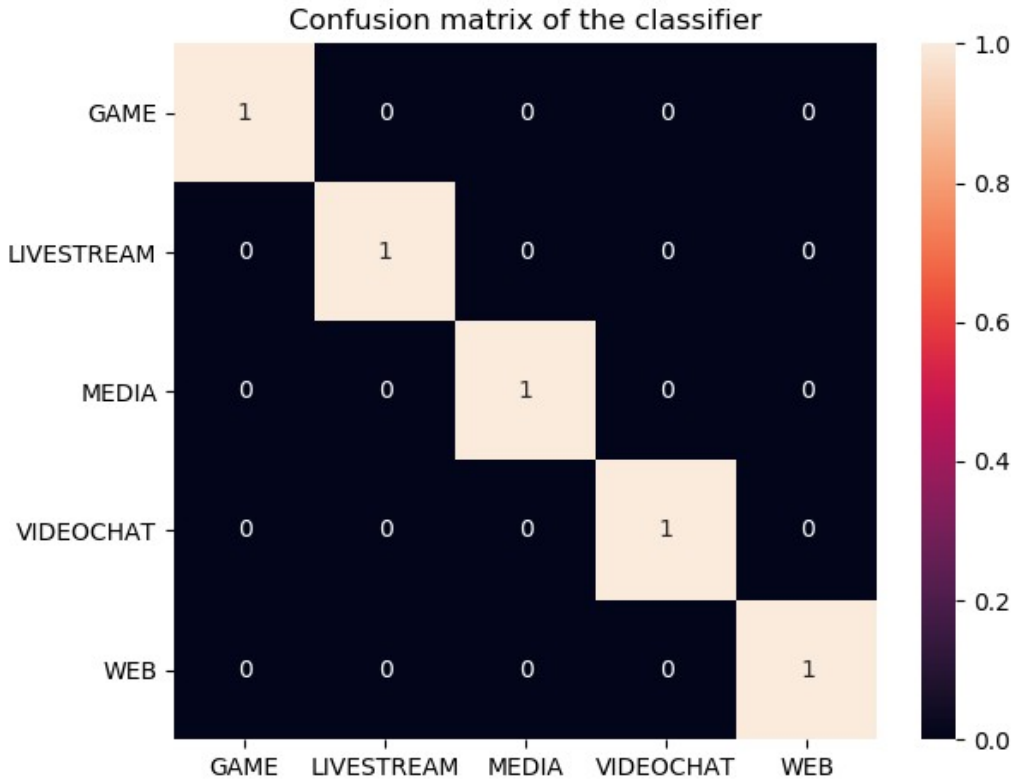

Figura 30: Matriz de confusión para Random Forest con 500 paquetes por clase.

Con 500 paquetes volvemos a tener un acierto del 100%.

Con 1000 paquetes:

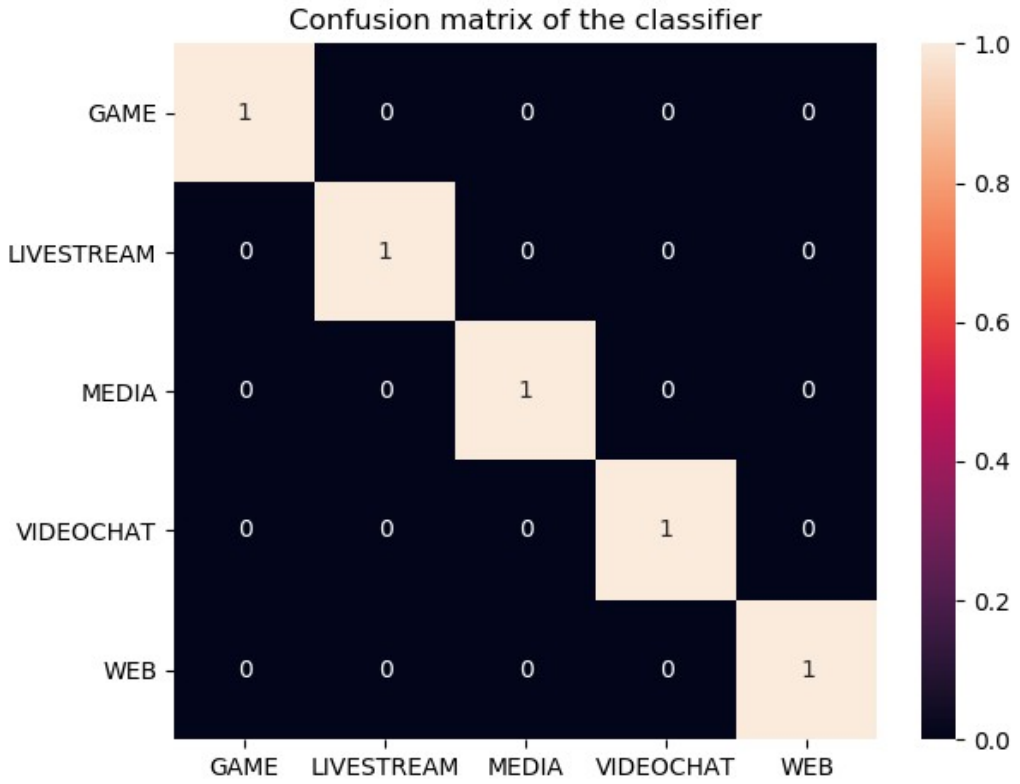

Figura 31: Matriz de confusión para Random Forest con 1000 paquetes por clase.

Con 1000 paquetes también tenemos un acierto del 100%.

Con 2000 paquetes:

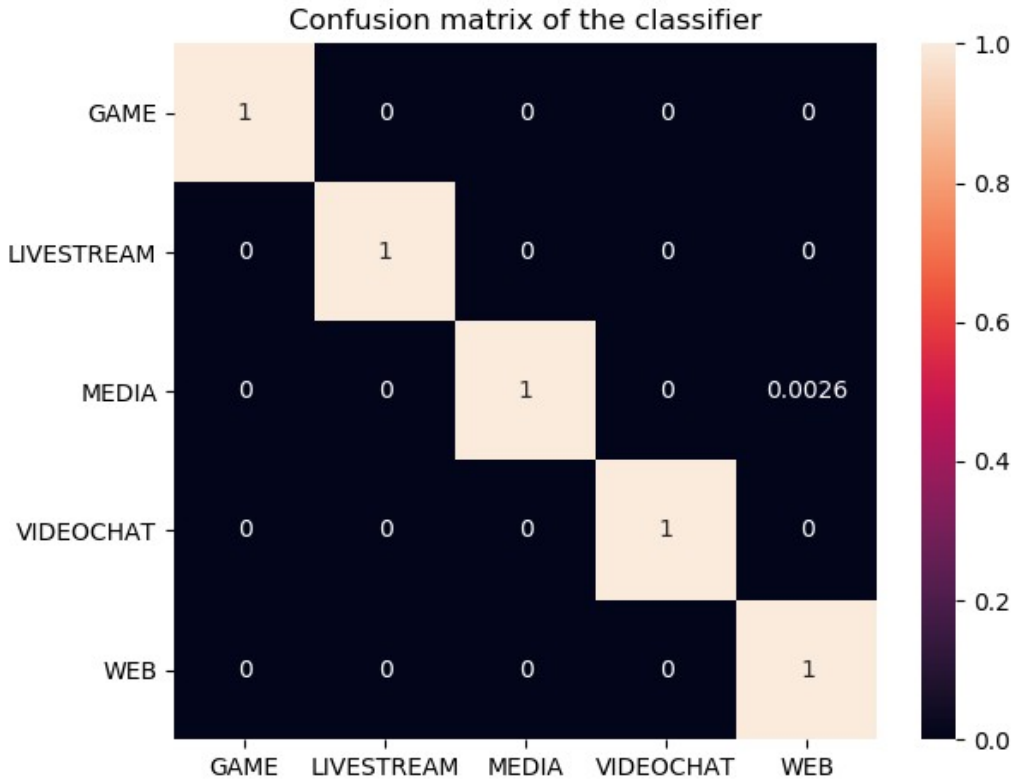

Figura 32: Matriz de confusión para Random Forest con 2000 paquetes por clase.

Con 2000 paquetes también tenemos un acierto del 100% aunque, en realidad, en el caso de la clase MEDIA podemos ver que clasifica incorrectamente el 0.26% como WEB, al ser tan poco (menos del 1%) se puede despreciar y por redondeo sale una tasa de acierto del 100%.

Con 5000 paquetes:

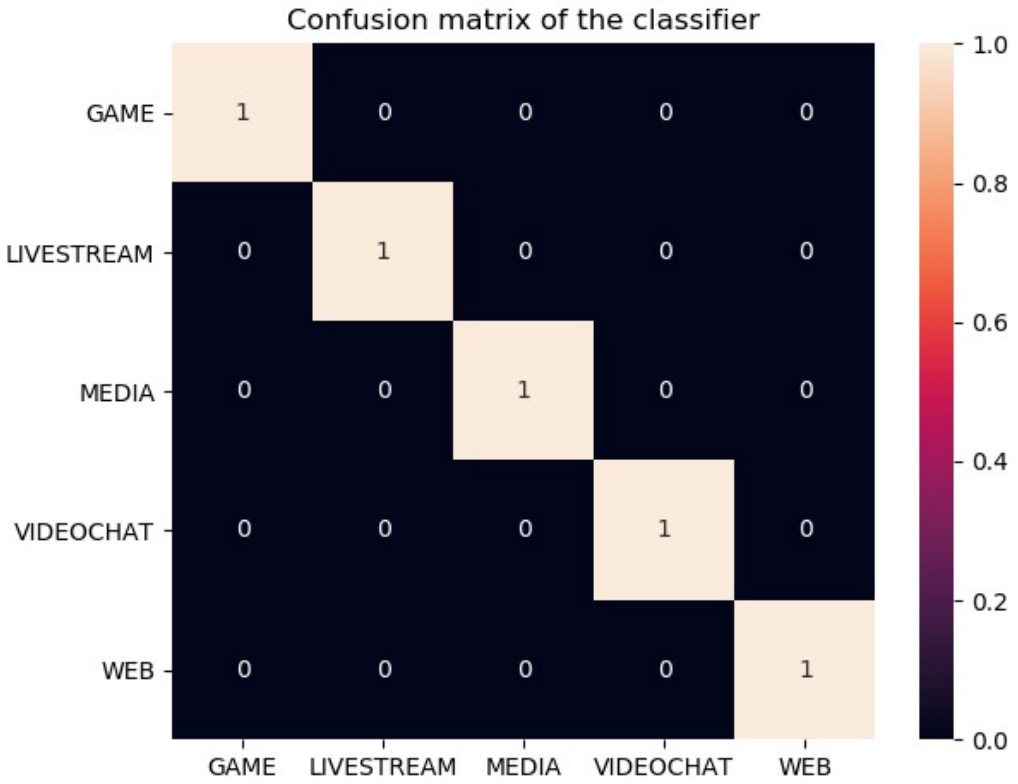

Figura 33: Matriz de confusión para Random Forest con 5000 paquetes por clase.

Con 5000 paquetes también sale una tasa de acierto perfecta (100%). Podemos apreciar que no es necesario entrenar la máquina con tantos paquetes ya que con 100 ya obteníamos resultados perfectos. Los resultados son prácticamente iguales a los obtenidos con Decision Tree y seguramente se deban a que usamos una muestra muy pequeña o que tenemos un tráfico muy concreto por lo que con tráfico real no deberíamos obtener resultados tan perfectos, sin embargo, deberíamos obtener mejores resultados con Random Forest que con Decision Tree por la naturaleza del algoritmo.

5.5. Resumen de resultados

A continuación, recogemos los resultados que hemos analizado individualmente en una sola tabla para poder analizarlos en su conjunto. De esta manera podremos concluir cuál de los algoritmos produce mejores resultados para el menor número de paquetes posible.

		Algoritmo		
		SVM	**Decision Tree**	**Random Forest**
Número de Paquetes	**100**	0.9372	1	1
	200	0.9121	1	1
	500	0.9313	0.9978	1
	1000	0.9636	0.9990	1
	2000	0.9565	0.9989	0.9994
	5000	0.9735	0.9996	1

Tabla 5: Tabla resumen de resultados

Como se puede apreciar en la tabla 5 los resultados son prácticamente perfectos, podemos ver cómo la precisión aumenta mientras más números de paquetes usamos en la muestra, es cierto que al principio Decision Tree y Random Forest dan resultados perfectos con una muestra pequeña pero esto es muy probable que se deba a que la muestra es demasiado pequeña y el resultado es afectado por el sobreajuste (overfitting). Al no bajar nunca del 90% estos resultados indican que cualquiera de estos algoritmos da buenos resultados aun con pocos paquetes. Además, podemos ver que el algoritmo con peores resultados es SVM y el algoritmo con mejores resultados es Random Forest. Estos resultados concuerdan con lo esperado ya que al usar un bosque, Random Forest debe dar mejores resultados que Decision Tree y aunque SVM da buenos resultados con predicciones binarias (donde tan solo existen dos clases o dos opciones) no parece dar tan buenos resultados con varias clases.

6. Conclusiones y trabajo futuro

A continuación concluimos el estudio. Describimos el trabajo realizado y los resultados obtenido y mencionamos posibles líneas de trabajo futuro por las que podría continuar el estudio. Finalmente, resumimos las contribuciones o aportaciones que ofrece este estudio, las fortalezas del trabajo.

6.1. Conclusiones y discusión

Para realizar este estudio se han tenido en cuenta estudios anteriores acerca la clasificación del tráfico, se ha podido comprobar el estado actual de las técnicas de clasificación: **basadas en el puerto**, **basadas en el payload**, **basadas en datos estadísticos del flujo**, **basadas en el host** y **basadas en grafos** [1]. Se ha comprobado cómo han ido evolucionando las técnicas de clasificación hasta la actualidad, viendo los problemas de las técnicas anteriores y cómo se han intentado ir resolviendo con nuevas técnicas. En resumen, las técnicas tradicionales ya no sirven debido a que las aplicaciones actuales no usan puertos registrados sino puertos dinámicos o los registrados para otras aplicaciones con la intención de disfrazar su tráfico (lo cual anula las técnicas basadas en el puerto) y usan encriptado y encapsulado (lo cual anula las técnicas basadas en el payload). De esta manera las aplicaciones intentan evitar filtros y firewalls [5]. Finalmente, los estudios actuales intentan usar algoritmos de Machine Learning para resolver estos problemas, por esta razón este estudio intenta comprobar la efectividad de 3 algoritmos de Machine Learning: **SVM**, **Decision Tree** y **Random Forest**. Estos estudios también nos han servido para comprobar qué parámetros suelen resultar más efectivos.

Se han implementado los 3 algoritmos de Machine Learning para clasificar tráfico de manera que se puedan estudiar los resultados de cada uno de ellos para distintas cantidades de paquetes de entrenamiento (100, 200, 500, 1000, 2000 y 5000). Para ello se han instalado las librerías necesarias (Scikit-learn, Scapy…) y se han implementado 3 scripts en Python para llevar a cabo los 3 pasos del estudio:

- **Preprocesado de las capturas** de Wireshark para extraer los datos que nos interesan en un fichero CSV.
- **División del fichero CSV** en varios con un distinto número de paquetes cada uno: 100, 200, 500, 1000, 2000 y 5000.
- **Clasificación de tráfico** usando los algoritmos de Machine Learning sobre los ficheros CSV, este script entrena los 3 algoritmos y comprueba la tasa de acierto, finalmente los resultados son expresados gráficamente para poder ser analizados.

Con las gráficas resultantes hemos podido comprobar que el algoritmo SVM no nos ha dado buenos resultados, aunque a partir de los 1000 paquetes empezaba a dar una alta tasa de acierto, seguía sin acercarse al 100% en todas las clases mientras que los otros dos algoritmos probados (Decision Tree y Random Forest) nos han dado resultados prácticamente correctos con tan solo 100 paquetes. Es cierto que en el algoritmo Decision Tree ha habido algún que otro fallo (aunque despreciable), por lo que podemos considerar que el algoritmo con mejores resultados ha sido Random Forest, sin embargo, ambos han dado resultados prácticamente perfectos y con muy pocos paquetes para el entrenamiento por lo que demuestra que ambos son más que válidos.

El estudio de la clasificación del tráfico resulta útil, por ejemplo, para un proveedor de servicios de Internet que quiere ofrecer un mejor servicio (filtro de spam y ataques maliciosos, calidad de servicio o QoS…) y, para ello, necesita poder analizar el tráfico que manejan. Mediante estos estudios se pretende comprobar la precisión de las técnicas actuales así como posibles técnicas futuras con el fin de descubrir una mejor forma de clasificar el tráfico.

Mejores técnicas de clasificación de tráficos son necesarias porque las técnicas actuales ya no son tan precisas, actualmente se ha estado usando el campo puerto que solía indicar a qué aplicación pertenece el tráfico debido a que un rango de ellos está asociado por la IANA a aplicaciones comunes. Sin embargo, por privacidad o

intentar evitar filtros del firewall, cada vez más aplicaciones usan el rango de puertos libre que la IANA deja usar para cualquier aplicación y, por lo tanto, es imposible saber a qué aplicación corresponden ya que cualquiera puede usarlos.

De esta manera, este proyecto puede resultar útil para comprobar la precisión de los algoritmos de Machine Learning estudiados (SVM, Random Forest y Decision Tree) usando varios parámetros (tamaño, protocolo, IP origen, IP destino, puerto origen, puerto destino y tiempo entre paquetes) obtenidos de la cabecera de los paquetes (ya que el contenido puede estar encriptado) o calculados a partir de la información que tenemos de ellos.

Resulta interesante saber si estas técnicas dan buenos resultados y comparar varios algoritmos para saber cuál es más óptimo. Un proveedor de servicios de Internet puede implementar el algoritmo que mejores resultados ha dado para clasificar el tráfico de la misma manera a la realizada en este estudio y una vez clasificado, podrá ofrecer un mejor servicio filtrando tráfico malicioso, dando preferencia a tráfico en tiempo real, etc.

6.2. Trabajo futuro

Tras realizar este estudio, podemos comprobar que las pruebas realizadas podrían ser más exhaustivas o se podrían usar trazas más grandes. El máximo número de paquetes usados han sido 5000 por clase debido a las limitaciones del hardware donde se ejecutaron las pruebas, sin embargo, como trabajo futuro, sería interesante para obtener resultados más concluyentes. En nuestro caso, los resultados obtenidos parecen ser demasiado perfectos debido a los pocos paquetes usados y lo bien diferenciados que son los paquetes de cada clase, sería útil ver los resultados usando paquetes de más de una aplicación por clase.

Las trazas usadas en este estudio se han obtenido manualmente para intentar obtener tráfico muy similar al tráfico de un usuario real, sin embargo, existen bases de datos como CAIDA (Center for Applied Internet Data Analysis) [34] que contienen un

conjunto de datos más amplio que los usados en este estudio y se podrían usar en una línea futura.

Teniendo los algoritmos de Machine Learning ya implementados, como trabajo futuro, también sería interesante usar estos algoritmos ya entrenados con paquetes en tiempo real que se pueden capturar con la librería scapy e intentar clasificarlos. Se podría crear una aplicación web que mostrase los resultados del tráfico clasificado en tiempo real así como un proxy o VPN de forma que en lugar de clasificar el tráfico originado en el ordenador donde se ejecutan los algoritmos, podamos clasificar el tráfico de otros dispositivos cuyo tráfico viaje a través del proxy.

Además, Machine Learning es un campo aún poco desarrollado en el ámbito de las redes que puede usarse en más casos a parte de clasificar tráfico. Otros estudios [35] [36] esperan poder usar Machine Learning para la administración de las redes y su orquestación, gestión de recursos de radio, gestión de movilidad y gestión de provisión de recursos. De esta manera, Machine Learning puede ayudar en la selección de parámetros y configuración de redes complejas así como encontrando patrones comunes de tráfico con tratamiento conocido que se puede automatizar o patrones desconocidos de los que puede avisar antes de que se produzcan problemas.

6.3 Contribuciones del proyecto

Para resolver el problema de la clasificación del tráfico de una red, este proyecto intenta usar la técnica de aprendizaje automático o Machine Learning. De esta manera, se han probado tres algoritmos (SVM, Decision Tree y Random Forest) que suelen dar buenos resultados para comprobar si es así según el número de paquetes con el que se entrenan.

Por lo tanto, hemos obtenido la relación entre la tasa de acierto y el número de paquetes y hemos podido comprobar que a pesar de que la muestra era pequeña, los algoritmos que dan mejores resultados han sido Decision Tree y Random Forest.

El número de paquetes a usar en el entrenamiento es importante ya que mientras más paquetes se usen en el entrenamiento, más se tarda en entrenar a la máquina y más

memoria se necesita para almacenar la información, por lo tanto es conveniente saber a partir de qué número de paquetes se dejan de obtener mejoras considerables para no perder el tiempo. Por esta razón, este trabajo comprueba qué algoritmos ofrecen mejores resultados según el número de paquetes usados en el entrenamiento.

REFERENCIAS BIBLIOGRÁFICAS

1: XUE, Yibo, WANG, Dawei and ZHANG, Luoshi, Traffic classification: Issues and challenges, 2013

2: MOORE, Andrew W. and ZUEV, Denis, Internet traffic classification using bayesian analysis techniques, 2005

3: EBEL, Jordan. Robust Streaming Video Traffic Classification.

4: YUAN, Ruixi, LI, Zhu, GUAN, Xiaohong and XU, Li. An SVM-based machine learning method for accurate internet traffic classification. *Information Systems Frontiers*. 2010, **12**(2), 149-156.

5: DAINOTTI, Alberto, PESCAPE, Antonio and CLAFFY, Kimberly C.. Issues and future directions in traffic classification. *IEEE network*. 2012, **26**(1), 35-40.

6: LIM, Yeon-sup, KIM, Hyun-chul, JEONG, Jiwoong, KIM, Chong-kwon, KWON, Ted Taekyoung and CHOI, Yanghee, Internet traffic classification demystified: on the sources of the discriminative power, 2010

7: NGUYEN, Thuy TT. and ARMITAGE, Grenville. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*. 2008, **10**(4), 56-76.

8: ANDERSSON, Ricky. *Classification of video traffic: An evaluation of video traffic classification using random forests and gradient boosted trees*. Master's thesis, Karlstad University, 2017.

9: SOYSAL, Murat and SCHMIDT, Ece Guran. Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison. *Performance Evaluation*. 2010, **67**(6), 451-467.

10: BUJLOW, Tomasz, RIAZ, Tahir and PEDERSEN, Jens Myrup, A method for classification of network traffic based on C5.0 Machine Learning Algorithm, 2012

11: MICHAEL, Ang Kun Joo, VALLA, Emma, NEGGATU, Natinael Solomon and MOORE, Andrew W.. *Network traffic classification via neural networks*. 2017. University of Cambridge, Computer Laboratory.

12:Wikimedia Foundation, Inc.. Pareto principle - Wikipedia [online]. 2018, [viewed 1 July 2018]. Available from: https://en.wikipedia.org/wiki/Pareto_principle

13:Scapy. Scapy [online]. 2017, [viewed 19 March 2018]. Available from: http://www.secdev.org/projects/scapy/

14:Tcpdump. TCPDUMP/LIBPCAP public repository [online]. 2018, [viewed 22 June 2018]. Available from: http://www.tcpdump.org/

15:Kaitai Project. Kaitai Struct: declarative binary format parsing language [online]. 2018, [viewed 7 March 2018]. Available from: http://kaitai.io/

16:Scapy. Scapy (Quick demo : an interactive session) [online]. 2017, [viewed 19 March 2018]. Available from: http://www.secdev.org/projects/scapy/demo.html

17:Google LLC.. TensorFlow [online]. 2018, [viewed 22 June 2018]. Available from: https://www.tensorflow.org/

18:Google LLC.. Get Started with Graph Execution | TensorFlow [online]. 2018, [viewed 22 June 2018]. Available from: https://www.tensorflow.org/get_started/get_started_for_beginners

19:scikit-learn developers. scikit-learn: machine learning in Python [online]. 2017, [viewed 7 March 2018]. Available from: http://scikit-learn.org/stable/

20:scikit-learn developers. About us — scikit-learn 0.19.1 documentation [online]. 2017, [viewed 7 March 2018]. Available from: http://scikit-learn.org/stable/about.html

21:Keras.io. Keras Documentation [online]. 2018, [viewed 7 March 2018]. Available from: https://keras.io/

22:Berkeley AI Research. Caffe | Deep Learning Framework [online]. 2018, [viewed 7 March 2018]. Available from: http://caffe.berkeleyvision.org/

23:Facebook. Caffe2 | A New Lightweight, Modular, and Scalable Deep Learning Framework [online]. 2018, [viewed 7 March 2018]. Available from: https://caffe2.ai/

24:LISA lab. Welcome — Theano 1.0.0 documentation [online]. 2017, [viewed 7 March 2018]. Available from: http://deeplearning.net/software/theano/

25:Torch.ch. Torch | Scientific computing for LuaJIT [online]. 2018, [viewed 7 March 2018]. Available from: http://torch.ch/

26: ZHANG, Jun, CHEN, Chao, XIANG, Yang, ZHOU, Wanlei and XIANG, Yong. Internet traffic classification by aggregating correlated naive bayes predictions. *IEEE Transactions on Information Forensics and Security*. 2013, **8**(1), 5-15.

27: ZHANG, Jun, XIANG, Yang, WANG, Yu, ZHOU, Wanlei, XIANG, Yong and GUAN, Yong. Network traffic classification using correlation information. *IEEE Transactions on Parallel and Distributed Systems*. 2013, **24**(1), 104-117.

28:scikit-learn developers. 1.4. Support Vector Machines — scikit-learn 0.19.1 documentation [online]. 2017, [viewed 27 June 2018]. Available from: http://scikit-learn.org/stable/modules/svm.html

29:scikit-learn developers. sklearn.svm.SVC — scikit-learn 0.19.1 documentation [online]. 2017, [viewed 1 July 2018]. Available from: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

30:scikit-learn developers. 1.10. Decision Trees — scikit-learn 0.19.1 documentation [online]. 2017, [viewed 27 June 2018]. Available from: http://scikit-learn.org/stable/modules/tree.html

31:scikit-learn developers. sklearn.tree.DecisionTreeClassifier — scikit-learn 0.19.1 documentation [online]. 2017, [viewed 1 July 2018]. Available from: http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

32:scikit-learn developers. 1.11. Ensemble methods — scikit-learn 0.19.1 documentation [online]. 2017, [viewed 27 June 2018]. Available from: http://scikit-learn.org/stable/modules/ensemble.html#forest

33:scikit-learn developers. 3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.19.1 documentation [online]. 2017, [viewed 1 July 2018]. Available from: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

34:CAIDA. CAIDA: Center for Applied Internet Data Analysis [online]. 2018, [viewed 10 March 2018]. Available from: http://www.caida.org/home/

35: LI, Rongpeng, ZHAO, Zhifeng, ZHOU, Xuan, DING, Guoru, CHEN, Yan, WANG, Zhongyao and ZHANG, Honggang. Intelligent 5G: When cellular networks meet artificial intelligence. *IEEE Wireless Communications*. 2017, **24**(5), 175-183.

36: JIANG, Chunxiao, ZHANG, Haijun, REN, Yong, HAN, Zhu, CHEN, Kwang-Cheng and HANZO, Lajos. Machine learning paradigms for next-generation wireless networks. *IEEE Wireless Communications*. 2017, **24**(2), 98-105.

ANEXOS

Instalación de librerías

Este proyecto está implementado en el lenguaje de programación **Python**, más concretamente la versión **3.6.5**, por lo que es necesario un intérprete compatible.

Las librerías de las que depende el proyecto son: **Scikit-Learn**, **Scapy**, **Pandas**, **Numpy**, **Graphviz**, **Matplotlib** y **Seaborn**.

Python 3.6.5

Un instalador de Python 3.6.5 para Mac/Windows puede ser descargado desde la página oficial:

https://www.python.org/downloads/release/python-365/

En cuanto a Linux, Python 3 viene preinstalado en algunas distribuciones (como Ubuntu), para aquellas en las que no esté preinstalado es bastante probable que esté disponible en los repositorios de la distribución, para distribuciones basadas en Debian bastaría con ejecutar el siguiente comando:

```
sudo apt-get install python3
```

Sin embargo, también es posible descargar el código fuente de la página oficial mencionada anteriormente y compilarlo manualmente.

Scikit-Learn

Esta librería contiene las implementaciones de los algoritmos de Machine Learning que usa el proyecto.

```
pip3 install -U scikit-learn[alldeps]
```

Scapy

Esta librería es usada para preprocesar las capturas realizadas con Wireshark.

```
pip3 install -U scapy-python3
```

Numpy

Esta librería implementa funciones matemáticas para trabajar con números y matrices, este proyecto la usa para trabajar con los datos que están organizados en matrices multidimensionales.

```
pip3 install numpy
```

Pandas

Esta librería es usada para leer archivos CSV y almacenarlos en memoria como Data Frames que pueden ser usados por Scapy.

```
pip3 install pandas
```

Graphviz

Esta librería se encarga de generar los gráficos del Árbol de Decisión a partir de los datos ofrecidos por Scikit-Learn una vez ha sido entrenado.

```
pip3 install graphviz
```

Matplot y Seaborn

Esta librería se encarga de generar los dibujos de las gráficas a bajo nivel, el proyecto usará Seaborn que se apoya en esta por lo que ambas son necesarias. Seaborn permite dibujar varios tipos de gráficas a partir de los datos de un Data Frame o una matriz de confusión.

```
pip install matplotlib seaborn
```

Ejecución del software

El proyecto está compuesto por 3 scripts Python que llevan a cabo distintas tareas:

- **pcap_to_csv.py**

  Se encarga del preprocesado de los datos, lee las capturas .pcapng generadas por Wireshark y genera el fichero CSV con los datos extraídos.

- **reduce_capture.py**

  Lee el fichero CSV y genera varios CSV con distintos números de paquetes [100, 200, 500, 1000, 2000 y 5000].

- **main.py**

  El script principal, se encarga de leer los ficheros CSV, ejecutar los 3 algoritmos de Machine Learning sobre esos datos y generar los gráficos con los resultados.

pcap_to_csv.py

Este script toma 3 parámetros: El fichero pcapng, el fichero CSV de salida y la etiqueta.

Si el fichero CSV ya existe se añadirá al final en lugar de sobre-escribirlo, por lo que se debe borrar antes de ejecutar el script si ya existiera y luego se podrá ejecutar el script para cada captura pcapng:

```
python pcap_to_csv.py captures/counter-strike.pcap
capture.packets.csv GAME
python pcap_to_csv.py captures/skype-videochat.pcapng
capture.packets.csv VIDEOCHAT
python pcap_to_csv.py captures/youtube-livestreaming.pcapng
capture.packets.csv LIVESTREAM
python pcap_to_csv.py captures/youtube-watch-videos.pcapng
capture.packets.csv MEDIA
python pcap_to_csv.py captures/web-browsing.pcapng
capture.packets.csv WEB
capture.packets.5000.csv
```

reduce_capture.py

Este script usa el primer argumento como parámetro csv_path para especificar el fichero CSV que se va a procesar. Se debe especificar la ruta del fichero sin incluir la extensión ".csv".

```
python reduce_capture.py capture.packets
```

Los ficheros resultantes serán almacenados en el mismo directorio donde se encuentra el fichero de entrada incluyendo al final el número de paquetes por cada etiqueta o clase:

```
capture.packets.100.csv
capture.packets.200.csv
capture.packets.500.csv
capture.packets.1000.csv
capture.packets.2000.csv
```

main.py

Este script toma como parámetros la ruta del fichero CSV a utilizar y la ruta de destino donde almacenar los gráficos generados.

```
python main.py capture.packets.100.csv out/100
```

En este caso utilizamos el fichero CSV que tiene solo 100 paquetes por cada clase y los resultados serán almacenados en el directorio out/100. De la misma manera podemos ejecutar los algoritmos sobre los otros CSV generados con el script reduce_capture.py:

```
python main.py capture.packets.200.csv out/200
python main.py capture.packets.500.csv out/500
python main.py capture.packets.1000.csv out/1000
python main.py capture.packets.2000.csv out/2000
python main.py capture.packets.5000.csv out/5000
```

Ejecución completa

La ejecución completa está descrita en un fichero **run.bat** para Windows y en un fichero **run.sh** para Linux/Mac. Sin embargo, no es recomendable ejecutarlo todo de una vez debido al largo tiempo que lleva todo el proceso completo, se pueden abrir los ficheros y ejecutar las líneas una por una en una consola de comandos.