Detección de Botnets usando Redes Neuronales

Datos del estudiante

- **Nombre:** Sergio
- **Apellidos:** Roselló Morell
- **DNI:** 53632974X
- **email:** Sergio-resello@hotmail.con

Información sobre el entorno

- **Sistema Operativo:** Arch Linux
- **Entorno de escritorio:** dwm
- **Versión de Python:** Python 3.8.2
- **Editor de texto:** NeoVim
- **Generación del documento:** Escrito en MD, compilado a LaTeX con

```
nnoremap <leader>e :! pandoc % -f markdown -t latex -s
-o %:r.pdf<cr>
```

Resumen

En el trabajo de investigación a continuación, se va a revisar el uso de redes neuronales para catalogar y detectar Botnets en la red. Los pasos que se realizan engloban de forma general cualquier problema de Machine Learning, en el que se obtiene el dataset, se trata, para que contenga los valores deseados, se entrena la red neuronal, se genera el modelo y se revisa, para luego refinar el resultado.

Indice

Planteamiento

Descripción y planteamiento del problema

El problema que se plantea en este análisis es identificar trafico de red malicioso en una red a tiempo real.

La solución debe poder analizar a tiempo real los paquetes generados por la red y decidir si cada paquete individual es o no un paquete proveniente de un malware.

Este problema se adapta muy bien a una solución relacionada con Inteligencia Artificial. Mas en concreto, a modelos como las redes neuronales.

El principal inconveniente a la hora de solucionar un problema de selección en tiempo real con modelos tradicionales, aparece cuando se pretende analizar el trafico y compararlo con varios ejemplos de malware. Ya sea comparando directamente pequeños atributos, como el paquete de red entero, al final, estamos comparando con características que ya conocemos e identificamos como maliciosas. Esto quiere decir que estamos reaccionando al problema, no tomando medidas pro-activas al problema en quistión. La finalidad del problema es ser capaces de detectar malware, aunque no se haya detectado anteriormente el tipo especifico de malware siendo analizado.

Solucionar el problema en cuestión con métodos tradicionales, como bien puede ser comprobaciones secuenciales de características del paquete de red a analizar incrementa rápidamente la complejidad del algoritmo. Ademas, cada vez que se detecten nuevos casos, se debe integrar la comprobación al programa. Esto hace que sea imposible mantenerlo actualizado.

Existen otras técnicas, no tan primitivas, como por ejemplo comprobación de hashes, tanto completos, como parciales del paquete de red. Si podemos identificar las características comunes en los paquetes de red enviados entre el malware y el servidor C&C, podemos cifrar estos datos en hashes, que se revisaran contra los paquetes de red a medida que pasan por la red.

Descripción de los datos a utilizar

Para desarrollar un modelo de detección de paquetes maliciosos, lo mas importante es la calidad de los datos iniciales que tenemos. Si generamos el modelo con datos buenos, el modelo puede inferir, en muchas ocasiones el trafico de red maligno.

Es muy importante tener unos datos tanto específicos, como generales, con distintas muestras y combinaciones de paquetes de red malignos, ya que estos son la parte mas critica del proyecto.

Como se ha mencionado superficialmente en la sección anterior, un modelo de detección basado en una red neuronal esta preparado para operar en un entorno de producción, en el que los paquetes de red que va a revisar no son exactamente

iguales que con los que ha entrenado, de esta forma, entrenando con un conjunto de datos lo suficientemente rico, podemos inferir la clase del paquete de red.

En el caso del entrenamiento del modelo, vamos a seleccionar un dataset realista, que contenga tanto trafico de red benigno como maligno y varios ejemplos de cada tipo.

Selección de Dataset de entrenamiento Se ha usado la pagina web de www.secrepo.com para buscar un dataset que contenga las propiedades deseadas para el estudio.

Las cualidades que se necesitan en el dataset son:

- Trafico de malware hacia servidores C&C
- Trafico de aplicaciones no maliciosas
- Cantidad de información (Necesario para poder inferir comportamientos y generar modelos de datos)
- Calidad de información (Ejemplos de trafico claro)

Siguiendo los requisitos demarcados anteriormente, se han encontrado varios datasets, entre estos, se va a usar `ISOT HTTP Botnet Dataset`, desarrollado por Alenazi A y compañeros para una charla con titulo: "Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments".

Este dataset ha sido generado por la universidad de Victoria en el año 2017 y consiste en nueve capturas de trafico malicioso y 19 capturas de trafico de aplicaciones no maliciosas, como por ejemplo Dropbox o Avast.

Cada registro se puede trazar directamente a una IP que contiene únicamente el programa a analizar, ya sea malware o no. Esto quiere decir que el dataset es claro, de forma que se puede analizar de forma manual y llegar a una serie de conclusiones desde el primer momento. Esto nos permite deducir los posibles resultados del análisis con el algoritmo de aprendizaje automático.

Es importante que tanto los paquetes maliciosos como los corrientes se hayan capturado al mismo tiempo ya que es una de las formas que tenemos para generar un fragmento de entrenamiento y test verídico.

Una de las ventajas de este dataset frente a otros que también cumplían los requisitos es que los datos vienen en un formato `.pcap`. Este detalle permite al investigador tomar el control de la información que se va a añadir al `.csv` para proporcionar al programa de generación del modelo.

Datos sobre los que opera el modelo en producción Una vez este el modelo terminado, va a ser puesto en un punto estratégico de la red, en el que tiene visibilidad de los paquetes entrantes y salientes de la misma. Un sitio en el que podría estar es en el router o switch de la red, actuando de firewall.

Cuando este desplegado, este sistema revisa cada paquete que entra o sale de la red y avisa (En caso de IDS) o ejecuta medidas preventivas (En caso de IPS)

según el tipo de paquete que detecte.

En principio, si el modelo ha sido entrenado correctamente, no hace falta volver a entrenarlo con mas datos, pero si lo ponemos y vemos que no detecta correctamente el tipo de paquete que analiza, puede que tengamos un problema de especificidad de datos. El dataset que hemos seleccionado para entrenar el modelo es bastante especifico, tiene datos concretos de aplicaciones concretas, como Dropbox, pero no de trafico realista de red, como usuarios buscando cosas en Google o otras aplicaciones diversas. En caso de que el modelo no detecte correctamente los paquetes maliciosos es obtener una muestra de nuestra propia red y analizar su trafico, junto con muestras de botnets.

Posibles algoritmos de aprendizaje automático a usar

En la asignatura cursada, se han realizado estudios sobre algoritmos de 'clustering', Clasificación, Probabilidad y 'Del Learning'

Teniendo en cuenta los algoritmos que se han aprendido durante el curso, se decide ahondar mas en los recursos sobre `Deep Learning`.

Los motivos por esta selección, entre otros son:

- Modelo adaptado al problema
- Preferencia personal

Breve comparación de los algoritmos estudiados TODO

Hay dos grandes categorías de redes neuronales:

- RNN (Recurrent Neural Networks)
  - LSTM (Long Short-Term Memory)
- CNN (Convolutional Neural Networks)

Por que Redes neuronales Entre los distintos tipos de redes neuronales existentes, se ha optado por entrenar el modelo con <++>

Gestión de datos (Entrenamiento, modelado, normalización, categorización) Hay una serie de cambios que son necesarios hacer cuando se adapta un dataset de datos crudos a un dataset valido para ser la entrada de un modelo.

Los pasos que se van a tener que seguir son:

- Seleccionar *features* con las que nos quedamos
- Convertir datos a `csv`
- Sanado de el dataset
- Determinar datos categóricos y continuos
- Codificar datos categóricos y continuos
- Dividir el dataset en datos de prueba y entrenamiento

Elección de 'features' Una de las decisiones mas importantes cuando seleccionamos un dataset es saber que los datos están disponibles de la forma mas pura posible. Esto hace que sea mas complicado trabajar con ellos desde un inicio, debido a que se tienen que convertir y mutar para que sirvan como datos de entrada al modelo, pero la ventaja que tienen es que no son específicos, es decir, los mismos datos, pueden servir para solucionar distintos problemas.

En nuestro caso, al tener acceso a los datos en formato `pcap`, podemos decidir que datos nos interesan, de entre una gran variedad de posibilidades.

Conversión de datos a csv Los algoritmos de aprendizaje automático como las redes neuronal usan una matriz como datos de entrada. Una de las formas mas similares a las matrices, son las tablas `csv`, en la que la relación es evidente. Se van a tener que mutar los datos crudos a datos con formato `csv`, que luego leeremos con `python` para importar a nuestro modelo.

Sanado de dataset Es posible que el dataset generado tras convertir los datos de `pcap` a `csv` contenga errores. Un ejemplo de error es que cualquier campo no tenga valor. Si se detecta este error, debemos arreglarlo, ya que para las redes neuronales, es mejor que todos los campos tengan un valor. En este caso, el valor nulo lo cambiamos a `UNKNOWN`. Todos estos errores deben ser contemplados y arreglados.

En este dataset en concreto, los pasos que hemos tenido que hacer para sanar el dataset son:

- Eliminar las columnas extra introducidas por la herramienta de extracción de "features"
- Eliminar los valores nulos en el dataset
- Eliminar las tuplas que contienen información corrupta, en este caso, ha habido un problema con la herramienta de extracción de features `tshark`, en la que ha introducido dos veces las columnas de `ip.src` e `ip.dst`.

Los pasos de sanado se encuentran en el script `sanitize.py` incluido en la entrega de la practica.

Determinar datos categóricos y continuos Las variables categóricas son las que pueden obtener valores concretos, predefinidos dentro de una serie de posibilidades.

Las variables continuas, son las que pueden obtener valores infinitos, es decir, existe una infinidad de valores continuos. Estas variables suelen ser numéricas.

En nuestro caso, las variables categóricas son:

- `ip.src`
- `ip.dst`
- `_ws.col.Protocol`
- `_ws.col.Info`

La variable continua que tenemos es:

- `frame.len`

Codificar datos categóricos y continuos Tanto las variables categóricas como las continuas, se deben codificar de una forma especifica para que la red neuronal interprete los valores correctamente.

Las formas mas populares de codificar las variables categóricas es con encodeado `one-hot`, mientras que las continuas, se pueden encodear normalizando sus valores.

Dividir el dataset en entreno y test Es buena practica subdividir el dataset en dos.

Estas dos secciones serán la sección de entreno y la sección de prueba. El modelo se entrena con la sección de entreno, pero se reserva una sección, por lo general del 30% del tamaño del dataset para revisar el modelo al acabar la fase de entreno. Una de las razones por las que se hace esto, es para asegurarnos que nuestro modelo funciona correctamente. Esto es porque hemos usado unos datos para entrenar a nuestro modelo, pero estos datos, ya los ha visto, y el modelo se ha configurado de acorde con estos. A nosotros nos interesa generar un modelo que sea capaz de decidir con datos que no se hayan usado nunca, ya que esta es su finalidad.

Resultados esperados del proceso

Cuando se acabe de entrenar el modelo, este nos proporciona un porcentaje de acierto, basado en los datos de test. Este sera siempre nuestro limite superior de probabilidad de acierto. Una vez estemos satisfechos con este valor, si satisface nuestros requisitos a nivel de red, podremos desplegar el modelo.

Una vez despleguemos el modelo en nuestro entorno de producción, debemos estar al corriente de los paquetes de red que marca como `Botnets`. Si implementamos un IDS, podemos programar un servicio que nos notifique mediante correo electrónico o incluso por un canal de Slack, una plataforma de gestión de trabajo.

Implementación

Programas utilizados

Para realizar este ejercicio, se ha contado con una plataforma creada por Google específicamente para realizar proyectos de estas características. Una vez realizado, se ha descargado y ejecutado en mi maquina local, para asegurar que todas las fases funcionan correctamente.

La limitación de la plataforma de Google es que, al usar la capa gratuita, te proporcionan una maquina con poca RAM y procesador. Por este motivo, se han tenido que hacer algunas modificaciones desde la descarga del archivo de

la plataforma de Google a la plataforma local. Uno de los mas evidentes, es el dataset. A la plataforma, estaba subiendo una muestra pequeña del original. Ahora, esto lo hago directamente desde el código.

En remoto Se usa un framework hecho por Google llamado colaboratory, que deja todo preparado para realizar análisis de datos para Aprendizaje Automático. Tiene la forma de los cuadernos Jupiter y permite tener código y texto en una misma vista.

En local, Programas/Ayudas utilizadas En esta practica, se han usado varias tecnologías. Las mas importantes son:

- Python
- Keras con TensorFlow
- Bash

En relación a Python, podíamos usar la versión 2 o 3 del mismo lenguaje de programación. Se ha decidido usar la versión 3 ya que es lo recomendado por la comunidad.

En relación a Keras, este es el framework mas utilizado para codificar modelos de redes neuronales. Se describe su elección en mas detalle en la siguiente sección.

En relación a Bash, es un lenguaje de scripting muy potente. Funciona como una capa de unión entre los programas que el usuario tiene incluidos en su "Path". Una de las utilidades mas importantes del mismo es la posibilidad de vincular la salida de datos de un programa directamente con la entrada de datos del siguiente. Esto permite al usuario crear cadenas de flujo de datos de una forma muy sencilla y eficiente.

Elección del framework

A la hora de generar el modelo, nos encontramos con varias opciones para llegar a la misma finalidad. Entre las opciones, tenemos:

- SciKit, un framework de aprendizaje automático
- PyTorch, un framework de aprendizaje profundo
- Keras, un framework de aprendizaje profundo

Viendo las opciones anteriores, decidimos usar Keras, debido a su versatilidad y abstracción de los algoritmos de aprendizaje profundo. Este framework nos puede proporcionar la potencia de varios frameworks de redes neuronales como TensorFlow, Theano o CNTK. Nosotros vamos a usar Keras en combinación con TensorFlow para generar el modelo.

Mas concretamente, vamos a usar el modelo `sequential_model` para generar nuestra red neuronal.

Importación del dataset a Python En este dataset conviven tanto datos categóricos como continuos.

Se ha tomado la decisión de no incluir los datos del `epoch_date` ya que no van tienen mucha relevancia a la hora de generar la red neuronal.

Los datos son categóricos, debemos cambiarlos a datos `dummy` antes de que los use el algoritmo de generación del modelo. Para realizar este paso, usamos la codificación `one-hot`.

Los datos continuos, debemos normalizarlos, para eso, tenemos que asignar 0 al valor mas bajo y 1 al valor mas alto. Una vez tengamos el dataset tratado para ser ingerido por el modelo de la red neuronal, podemos empezar a entrenar, con un 70% del dataset, para posteriormente revisar con un 30% del dataset.

A excepción de la fase de tratado de datos y preparado del dataset, todas la demás fases, están en el archivo llamado: `modeloSecuencial.py`.

Los pasos que se toman para generar el dataset son:

Leer el `.csv` En este paso, se usa el método `read_csv` de `pandas`, pasándole los tipos de datos que se va a encontrar en cada columna, para optimizar mas la carga del dataset. Ademas, se le dice al método el numero de linea en el que se encuentra el nombre de cada columna.

Preparar los datos de entrada para el modelo En esta sección, se codifican los datos categóricos y continuos en una matriz que el modelo es capaz de entender y utilizar.

Los métodos que se han usado para cifrar los datos son:

- *OneHotEncoder* de sklearn.preprocessing
- *Normalize* de sklearn.preprocessing

Durante la codificación del script, uno de los errores que había cometido era dividir el dataset en datos de entrenamiento y testeo antes de preparar el dataset. El error con el que me estaba encontrando era que el tamaño del dataset de entrada para revisar el modelo no era el mismo que el tamaño que el dataset de entrenamiento del modelo, por tanto, no se podía revisar el modelo con los datos de prueba.

Al darme cuenta de este fallo, se convierten previamente todos los datos de entrada al modelo antes de subdividir el dataset en datos de entrenamiento y datos de testeo.

Pasos de ejecución

Vamos a establecer el directorio principal desde el cual trabajaremos durante todo el ejercicio. De ahora en adelante, esta sera la carpeta base de esta practica. (`~/`) Este se llama:

`TrabajoFinal`

1. Configuración del espacio de trabajo Descargamos el dataset desde la siguiente URL: ISOT HTTP Botnet Database a nuestro directorio raíz.

Extraemos el dataset en este directorio, de forma que se crea un directorio llamado `isot_app_and_botnet_dataset`.

2. Tratado de datos Copiar los scripts:

- `extraction.sh`
- `sanitize.py`
- `identifyAndsort.sh`
- `prepareData.sh`

Al directorio llamado `isot_app_and_botnet_dataset`.

Una vez tenemos estos archivos en nuestro directorio, procedemos a **ejecutar el `script extraction.sh`.** Este script convierte todos los datos de `pcap` a `csv`. Ademas, concatena todos los datasets con paquetes maliciosos en un dataset que contiene únicamente paquetes maliciosos y todos los datasets con trafico legitimo en un solo dataset que contiene únicamente paquetes legítimos.

Seguimos **ejecutando el script `sanitize.py`**, que elimina las comas extra que contiene el campo `_ws.col.Info` y ademas, rellena los valores nulos con el valor *UNKNOWN* y revisa el archivo en busca de entradas corruptas.

En el caso de nuestro dataset, el programa `tshark` genera muchas entradas corruptas relacionadas con la dirección `IP 192.168.50.17`. El error se puede detectar analizando el campo `frame.len` en busca de una dirección `IP`. Algunas formas en las que se hubiese podido tratar las entradas corruptas son:

- Eliminado
- Acondicionado/Arreglado

En nuestro caso, analizando las entradas corruptas, se puede observar que cada una contiene los campos `ip.src` e `ip.dst` duplicados. Eliminado esta duplicidad, se hubiese podido contar con estas tuplas.

Se ha decidido eliminar las ocurrencias corruptas, debido a que tenemos mucha variedad de datos.

Al terminar la operación anterior, añadimos una columna extra a cada uno de los datasets, para que el modelo pueda identificar que paquete de red es legitimo y cual es *Botnet*. Ademas, unimos y ordenamos por tiempo los paquetes de red. Esto se hace **ejecutando el comando identifyAndsort.sh**

Como alternativa, se ha preparado un script que automáticamente genera los archivos indicados para importar con `Python`. **Ejecutando prepareData.sh**. Es necesario tener los requisitos necesarios por `Python` para ejecutar el script

de `Python`. Estos están en el archivo `requirements.txt`. Para **instalar los requisitos, se ejecuta el comando `pip install -r requirements.txt`**

Limitaciones/restricciones en la implementación

Este modelo es capaz de discriminar los paquetes de red maliciosos de los comunes teniendo en cuenta el grupo reducido de paquetes que se ha usado. A día de hoy, no se puede asegurar el funcionamiento del modelo con datos o programas maliciosos cuyo trafico no se ha capturado y utilizado para generar el modelo. Dicho esto, es probable que el modelo pueda inferir en mayor o menor medida trafico no revisado anteriormente, ya que es justo el dominio de las redes neuronales.

Una de las limitaciones a la hora de realizar esta implementación ha sido la ingente cantidad de datos que contiene el dataset. El archivo de texto plano que contiene todos los paquetes del dataset tiene en torno a diez millones de entrada, esto son en torno a 10 millones de datos sobre paquetes de red. Para analizar todos estos datos, al menos de la forma en la que se ha procedido en esta ocasión, se hubiese necesitado un ordenador muy capaz, con mas de siete TiB de memoria.

Pruebas y resultados

Proceso a seguir para obtener los datos

1. Descarga de los archivos desde mi repositorio de GitHub: sergiorosello o desde la entrega proporcionada a través de AlF.
2. Descarga del dataset desde ISOT HTTP Botnet Database
3. Ejecutar el script proporcionado en la entrega llamado `prepareData.sh`.

Al llegar a este punto, se habrá generado un archivo llamado `sorted_merged_traffic.csv`. El programa de Python en el que se encuentra el modelo se encarga de extraer una muestra de los datos dentro del dataset, ya saneado.

Aplicación de los programas/scripts

El script en el que se define el modelo y se entrena se llama `modeloSecuencial.py`.

Para **ejecutar este archivo, se puede usar el comando `python modeloSecuencial.py`**

Este script se encarga de:

- Leer una fracción del dataset de forma optimizada
- Separar el nuevo dataset en datos de entrada y salida
- Preparar los datos de entrada (Tanto los de prueba como los de entrenamiento)
- Subdividir el dataset en datos de entrada de prueba y entrenamiento y datos de salida de prueba y entrenamiento

- Definir el modelo
- Definir las capas de la red neuronal
- Compilar el modelo
- Entrenar el modelo
- Evaluar el rendimiento del modelo

Automáticamente, se encodea el dataset para poder usarse con el modelo, se genera el modelo, se entrena el mismo y se calcula el porcentaje de aciertos basándose en los datos de prueba.

Casos comprobados y valores de datos iniciales y de parámetros

Modelo Se ha usado el modelo Secuencial, ya que proporciona la flexibilidad que se necesita y la facilidad de uso. En esta primera practica, me ha parecido interesante empezar desde la base.

En futuras implementaciones, sin duda empezare a usar el API Funcional.

Función de activación Se han usado dos funciones de activación desde el principio de la practica. Las funciones de activación determinan si una neurona cumple con los requisitos para activarse. Si sobrepasa la función de activación, esta neurona se activa y el flujo continua a través de esa neurona por la red neuronal.

Los dos tipos de funciones de activación que se han usado son:

- `relu`
  - Es una mejora a la función de activación binaria.
  - Es sencilla de aplicar, por tanto es rápida, buena para una primera capa
  - Como desventaja, puede generar neuronas "muertas"
- `sigmoid`
  - Devuelve siempre un valor entre 0 y 1
  - Valores gradualmente mas altos, devuelven valores mas próximos a 1 y viceversa

Optimizadores Existen siete optimizadores distintos que el analista puede usar a la hora de compilar el modelo de red neuronal. Entre estos, el optimizador que se ha decidido usar es `adam`, debido a que según kingma et al., 2014 "Es computacionalmente eficiente, no necesita mucha memoria, invariante al re-escalado diagonal de gradientes y es adecuado para problemas que tienen muchos datos/parámetros."

Métricas Cuando se ha compilado el modelo, se ha usado la métrica "accuracy" para saber el numero de veces que el modelo predice el "label". Esto es: Saber la frecuencia con la que el modelo acierta determinando el tipo de trafico que se ha comprobado.

Resultados obtenidos e interpretación de los datos

En la primera prueba que se han realizado:

- **Modelo:** Secuencial
- **Capas:**
  - Dense, 32 neuronas, activación relu, kernel_initializer he_normal
  - Dense, 1 neurona, activación sigmoide
- **Compilacion:**
  - Loss binary_crossentropy, optimizador adam, metrics accuracy
- **Entreno:**
  - Epochs 10
  - batch_size 16

Se ha llegado a una precisión del 96% en el ultimo 'epoch' del entrenamiento, pero se ha obtenido un 94% de acierto usando el modelo con los datos de test. Estos datos son bastante buenos.

A partir de estos datos se ha procedido a cambiar los parámetros de la red neuronal para ajustarla mejor. (Desde función de activación, Optimizadores, nuevas capas, de distintos tipos)

El máximo porcentaje de acierto que se ha conseguido ha sido con el modelo presentado.

TODO: Cambiar los datos del modelo presentado

- **Modelo:** Secuencial
- **Capas:**
  - Dense, 32 neuronas, activación relu, kernel_initializer he_normal
  - Dense, 1 neurona, activación sigmoide
- **Compilacion:**
  - Loss binary_crossentropy, optimizador adam, metrics accuracy
- **Entreno:**
  - Epochs 10
  - batch_size 16

Otros valores a revisar

En esta practica se ha usado el modulo `sequential_model`, debido a que es una red neuronal sencilla, pero seria muy interesante entrenar el modelo con modelos mas avanzados, que contengan neuronas compartidas entre capas, distintas, entradas y salidas por capa o grafos de capas.

La fase de elección y tratado de datos de entrada para el modelo es una de las mas importantes a la hora de realizar un buen modelo. Por este motivo, me hubiese gustado poder variar los datos de entrada. Como ejemplo, en vez de usar un extracto proporcional de datos de entrada (Teniendo en cuenta que hay mucho mas trafico de botnets que convencional) usar datos de entrada que

contengan el mismo numero de paquetes de Botnet como de paquetes de red convencionales.

Creo que de esta forma, el modelo hubiese tenido mas variedad de información. Quizá, una posible consecuencia es esto es que hubiese salido mas general. Entiendo que no, porque los datos que se han seleccionado del dataset, no han sido de un mismo Botnet, sino un extracto de todos los datos. De todas formas, me quedo con la duda.

Posibles mejoras

El dataset ha sido generado en una misma red. Cada maquina ha estado enviando trafico especifico de un malware concreto. Esto es beneficioso, porque podemos identificar a simple vista (Según la IP) si ese trafico es malicioso o no. El inconveniente que introduce este método es que no es trafico de red verdadero, ya que en la vida real, una sola maquina genera trafico tanto normal como malicioso.

Otro inconveniente de la forma en la que se ha capturado el dataset es que los datos malignos se han capturado antes que los normales, haciendo que, si se ordena todo el dataset, todo el trafico de red este segmentado por naturaleza.

Un problema persistente a lo largo de toda la practica ha sido la enorme cantidad de datos disponibles. Desde la adquisición de los mismos, pasando por su gestión, hasta su utilización con el modelo de red neuronal.

Comentarios sobre la realización de la actividad

La actividad ha sido un ejercicio completo de tratado de datos para un propósito concreto. Ha sido muy interesante pasar por todas las fases a las que se enfrenta un analista de datos Sobre todo, darme cuenta que el manejo de datos y su correcta codificación consumen mas tiempo incluso que implementar el modelo. Mas aun si el analista tiene claro que modelo debe implementar para solucionar el problema. En mi experiencia, esta parte ha sido la mas complicada.

Bibliografía

- *Neural networks*:
  - `https://scikit-learn.org/stable/modules/ neural_networks_supervised.html`
- *Dataset*:
  - Alenazi A., Traore I., Ganame K., Woungang I. (2017) Holistic Model for HTTP Botnet Detection Based on DNS Traffic Analysis. In: Traore I., Woungang I., Awad A. (eds) Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. ISDDC 2017. Lecture Notes in Computer Science, vol 10618. Springer, Cham
- *one_not encoding*

  - https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEr
- *Categorical_functions*
  - https://keras.io/api/utils/python_utils/#to_categorical-function
- *read_csv*
  - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html
- *Keras sequential model*:
  - https://keras.io/guides/sequential_model/
- *Keras training and evaluation*:
  - https://keras.io/guides/training_with_built_in_methods/
- *Keras functional model*:
  - https://keras.io/guides/functional_api/