Actividad 1 de evaluación continua a distancia para la asignatura: "Introducción al Aprendizaje Automático para Ciberseguridad"

curso 2019-20

1. Información previa

Es importante que en primer lugar **lea todo este documento completo detenidamente** y hasta el final, ya que proporciona paso a paso toda la información necesaria para realizar la actividad. También es necesario haber leído previamente el Manual Didáctico en el curso virtual de la asignatura y haber seguido las recomendaciones de estudio incluidas en el mismo.

1.1. Resumen

Se propone la realización de una práctica guiada sencilla basada en el código de ejemplo proporcionado con el libro base que se indica en el Manual Didáctico. En este caso se trata de una práctica sobre el tema 2 de agrupamiento siguiendo el ejemplo dado en el libro base (capítulo 1) utilizando los ficheros de registro de accesos a un servidor web para intentar identificar las direcciones IP de posibles atacantes.

1.2. Forma de entrega

Esta es un tarea individual y personal de cada estudiante, no se permiten grupos. El resultado de la actividad debe redactarse en un documento, cuyo contenido y estructura se especifica detalladamente más adelante (sección 1.4), convertido a formato **PDF**. El documento se enviará únicamente a través de la tarea correspondiente **dentro del curso virtual** (en la página de "Actividades Evaluables" del menú izquierdo): `https://2020.cursosvirtuales.uned.es/dotlrn/posgrados/asignaturas/31109097-20/?page_num=3`

Por favor, **no** envíen la respuesta por ningún otro medio, ni por correo electrónico, para evitar confusiones, extravíos y problemas de gestión. Tampoco se aceptarán entregas fuera del plazo establecido para todos los estudiantes.

En caso de que desee añadir algún otro fichero muy relevante en su respuesta (código fuente, hoja de cálculo, etc.), debe empaquetar todos los ficheros juntos, incluido el documento PDF obligatorio, en un archivo comprimido compatible ZIP o RAR y subir ese archivo comprimido (en lugar del documento) a la tarea correspondiente dentro del curso virtual.

1.3. Plazo de entrega

La entrega se debe realizar subiendo el fichero correspondiente en la tarea del curso virtual **antes del día 10 de junio de 2020 a las 23:55h.** Se recomienda no esperar hasta el último día, para evitar las saturaciones[1] o problemas puntuales de acceso a internet, etc. Si surge algún problema para el envío, realice lo antes posible una consulta directamente al coordinador del equipo docente por email (pero no envíe el fichero por ese medio) a <jras@dia.uned.es>.

La respuesta de la actividad se puede enviar en el curso virtual tantas veces como se quiera hasta el plazo indicado, ya que solamente la **última versión** se guarda y es la única que se tendrá en cuenta para la evaluación al finalizar el plazo.

[1] Téngase en cuenta que entre los días 25 de mayo al 13 de junio de 9h a 20h. se realizarán en aLF (la plataforma de cursos virtuales) pruebas sincronizadas en otras asignaturas, lo que podría causar congestiones puntuales. Durante esos días se recomienda subir los ficheros fuera de ese horario (es decir, hacerlo preferiblemente después de las 20h y antes de las 8h.).

Se recomienda verificar que el fichero guardado en el servidor es el correcto, descargándolo después del envío para comprobar que es la versión correcta.

1.4. Contenidos y estructura del documento de respuesta

El documento de respuesta puede redactarse en cualquier editor, pero se debe **exportar a PDF** para enviarlo. Por ejemplo, se puede usar el editor de LibreOffice que permite exportar a PDF, es gratuito y está disponible para todas las plataformas (ver `http://libreoffice.org/` para obtenerlo). También es muy recomendable el programa LyX (ver `http://lyx.org/`) que puede generar PDF de calidad fácilmente y está disponible en la mayoría de distribuciones de Linux, y también para MS-Windows o Mac.

En el documento de respuesta (indicado en el apartado 1.2) se deben especificar los resultados de la actividad, y también las explicaciones de cómo se han obtenido éstos, ordenados según el siguiente **esquema**:

1. Datos del estudiante: Nombre, Apellidos, DNI o pasaporte y email de contacto. También se puede incluir opcionalmente un número de teléfono de contacto.

2. Información sobre el entorno y programas usados para esta actividad (incluyendo números de versiones): Sistema operativo, entorno de escritorio, versión de Python y editor de texto para generar el documento PDF. Adicionalmente se puede incluir información de otros programas (versiones de paquetes, etc.) o medios utilizados para la realización.

3. Breve resumen de la actividad a realizar.

4. Respuestas de cada uno de los apartados pedidos en la actividad (ver apartado 3). Es necesario separar y etiquetar claramente cada apartado y, aparte de la información o datos específicos requeridos, se recomienda incluir lo siguiente:

   a) Qué valores de parámetros se han usado para obtener la respuesta.

   b) Qué modificaciones de código se han hecho respecto a los originales.

   c) Qué fallos o errores se han encontrado y cómo se han solucionado.

5. Comentarios y opiniones: Dificultades o problemas encontrados, programas o ayudas utilizadas, comentarios sobre la realización de la actividad, etc.

6. Bibliografía o fuentes de información: Documentos, páginas web, libros, etc. consultados para la realización (con datos de título y editorial o URL de localización, etc.).

Es muy importante para la evaluación, seguir el esquema dado e **incluir explicaciones** de cómo se han realizado los cálculos (qué lista de datos, operación, resultado, etc.) para que se pueda calificar correctamente.

Utilice un formato sencillo y claro, con una redacción estructurada en apartados numerados en el documento. Por favor, no utilice imágenes ni formatos decorativos innecesarios, pues pueden valorarse negativamente.

Para incluir datos y resultados obtenidos en comandos de consola de texto, se debe copiar el texto y "pegarlo" como texto en el documento. Se le puede dar formato con un tipo de letra de paso fijo (*fixed font*) y preservar los saltos de línea[2]. No se deben insertar capturas gráficas de la pantalla de las ventanas de terminal de consola que ocupan bastante más y pueden tener peor calidad.

2. Instrucciones previas

Para la realización de la actividad, es necesario haber instalado y probado el código de ejemplo que se proporciona con el libro base indicado en el Manual Didáctico en el curso virtual de la asignatura. A continuación se indican los requisitos y pasos previos para conseguir que dicho código funcione adecuadamente.

[2]En el procesador LyX se puede insertar el texto dentro de un "Listado de Código" que preserva el espaciado.

2.1. Requisitos

2.1.1. Sistema operativo y paquetes necesarios

Los programas necesarios en esta actividad utilizan el código proporcionado con el libro base en el repositorio de GitHub correspondiente, por tanto es muy recomendable utilizar una sistema operativo Linux, preferiblemente una distribución derivada de Debian o Ubuntu reciente.

Los cálculos para la actividad se pueden realizar dentro de una máquina virtual con al menos 2GiB de memoria. Se pueden utilizar imágenes de máquinas virtuales (VDI o virtual disk image) ya preinstaladas de Debian, Ubuntu, Linux-Lite o cualquier otra desde osboxes.org. También hay muchas imágenes preinstaladas en virtualboxes.org y en descargarmaquinasvirtuales.com (en español). Las imágenes de disco de VirtualBox o de VMWare se pueden convertir también para usar en Parallels de Mac.

Para algunos comandos adicionales usados, será necesario tener instalados (con "apt" o la herramienta de paquetes correspondiente de la distribución de Linux utilizada) los paquetes siguientes: `wget`, `gzip`, `grep`.

Para la visualización de la proyección de datos (con Matplotlib), será necesario un entorno gráfico compatible X11, para lo cual sirve cualquier entorno de escritorio de los habituales en Linux (Gnome, KDE, XFCE, LXDE, etc.).

2.1.2. Entorno Python y módulos

Aunque el código de ejemplo utilizado en esta actividad está escrito originalmente en Python2, se recomienda utilizar Python3 para su realización, ya que las modificaciones necesarias para ello son triviales (tal como se indica al final del apartado 2.2.2).

Siempre que sea posible, se recomienda instalar los paquetes y bibliotecas de Python del sistema (con "apt" en sistemas Debian o Ubuntu, o bien la herramienta equivalente de la distribución Linux usada), e instalar únicamente con "pip3" aquellos módulos que no existan como paquetes del sistema.

Para esta actividad será necesario tener instalados los paquetes: `python3-sklearn`, `python3-matplotlib`, `python3-tk`, `python3-h5py`, `python3-requests`, `python3-dev`, `2to3`, con lo que se instalarán adicionalmente como dependencias otros paquetes necesarios como: `python3`, `python3-numpy`, `python3-scipy`, etc.

2.2. Ficheros necesarios

2.2.1. Código y datos de ejemplo

Los ficheros de código fuente necesarios se pueden descargar directamente del repositorio público GitHub[3] en un archivo comprimido (64.2MB): `https://github.com/cylance/IntroductionToMachineLearningFo archive/master.zip` que, una vez descomprimido (234.7MB), contiene el directorio: `IntroductionToMachineLearningForSecurityPros-master/clustering_example/` con los ficheros que se utilizarán en esta actividad.

2.2.2. Modificaciones del código

En el código fuente original, proporcionado en el repositorio GitHub del libro base, se han detectado algunos **errores que es necesario corregir**, además de adaptarlo para Python3, para que el código funcione correctamente. En esta actividad se utilizará el código de ejemplo del subdirectorio `clustering_example`, que se obtiene tal como se ha indicado en el apartado 2.2.1. Se deben hacer las siguientes modificaciones:

- Fichero: `clustering_example/vectorize_secrepo.py`
  - Editar la línea 87 para sustituir "`requests`" por "`responses`", de forma que quede así: `vectors[index, len(request_types) + ri] = v["responses"][r]`
  - Eliminar la línea 65, porque el código "`403`" está repetido.
  - Insertar una nueva línea después de la línea 31 (en su misma indentación) con el siguiente código: `response_code = int(response_code)`

[3] Alternativamente se puede clonar localmente el repositorio de GitHub con el comando:
`git clone https://github.com/cylance/IntroductionToMachineLearningForSecurityPros.git`

- Fichero: `clustering_example/visualize_vectors.py`
  - Editar la línea 17 para cambiarla a: `s=20,` que es el valor por omisión, para que los puntos no se dibujen tan gruesos.
- Conversión a Python3: para que el código sea ejecutable con Python3 es necesario modificarlo ejecutando, desde dentro del directorio `clustering_example/`, el siguiente comando[4]:
  `2to3 -w -x dict *.py`
  De esta forma se modificarán automáticamente los ficheros (mayormente cambios de `print()` y alguna envoltura `list()` de un iterador) y se crearán copias en ficheros `.bak` de los originales.

3. Resultados requeridos

En esta actividad se van a replicar los cálculos de ejemplo del libro base con los datos del repositorio GitHub, para analizarlos y compararlos con los resultados mostrados en el libro. También se propone realizar un análisis similar con otros datos diferentes para ver si es posible identificar grupos de IPs sospechosas.

3.1. Cálculos con los datos de ejemplo

En primer lugar, después de realizar la instalación y descarga de los ficheros tal como se ha indicado en el apartado 2, se deben ir ejecutando los mismos comandos (cambiando "`python`" por "`python3`", evidentemente) indicados en el apartado "*Clustering Session Utilizing HTTP Log Data*" del libro base (final del capítulo 1). Los scripts correspondientes se pueden encontrar en el directorio indicado en el apartado 2.2.1.

Como respuesta de este apartado (en el documento de la actividad explicado en el apartado 1.4), se deben indicar los resultados obtenidos en cada caso (*K-Means* y *DBSCAN*) y compararlos con los correspondientes del libro.

Es importante recordar que los resultados mostrados en el libro se corresponden con un subconjunto[5] de los datos incluidos en el repositorio GitHub, por lo cual es muy posible que los resultados sean algo diferentes numéricamente. El análisis y comparación que se pide se debe realizar de forma cualitativa para determinar si son coherentes y compatibles.

Avisos: En el texto del libro se indica que el análisis con *DBSCAN* se realiza, en primer lugar, usando los hiperparámetros: Eps=0.5 y MinPts=5, pero en el primer comando tiene argumento erróneo "`-m 2`", que debería ser "`-m 5`".
Después en el texto indica que incrementa Eps de 5 a 6, pero debería ser de 0.5 a 0.6 (ya que los vectores se han normalizado), lo cual se debería reflejar también en el argumento "`-e 0.6`" en el comando correspondiente.
Es recomendable revisar el código fuente del script `cluster_vectors.py` para comprender a qué corresponde cada argumento ("`-e`" y "`-m`") con los parámetros más importantes de la función `DBSCAN` (ver su documentación en `https://scikit-learn.org/`).

3.2. Cálculos con otros datos diferentes

También se pide realizar un análisis similar, pero aplicado a un conjunto distinto de ficheros de registro, para ver si sería posible identificar algún grupo (*cluster*) con potenciales direcciones IP peligrosas, pero sin tener la pista de direcciones sospechas como en el caso anterior.

Los datos incluidos en el repositorio GitHub dentro del subdirectorio `clustering_example/data/www.secrepo.org/self.logs/`, corresponden a los ficheros diarios de registro de acceso al servidor web `https://www.secrepo.com/` desde 2015 hasta mitad de 2017, que se encuentran públicamente disponibles en `https://www.secrepo.com/self.logs/`. Todos los ficheros tienen nombres

[4] En el comando se usa la opción "`-x dict`", que excluye las correcciones para envolver `dict.keys()` con `list()`, ya que en el único caso en estos ficheros no es necesario y perjudica la eficiencia.

[5] Incluso más pequeño que la restricción indicada en el texto, a las primeras 10000 IPs encontradas, que ya está implementada en el código fuente (ver líneas 36 o 37 originales de `vectorize_secrepo.py`). Aparte de que la función `os.listdir()` en Python no garantiza el orden de los ficheros, que depende del sistema.

de la forma forma: `access.log.AAAA-MM-DD` donde AAAA-MM-DD es la fecha correspondiente (año, mes, día).

En este apartado se trata de analizar únicamente los datos en los ficheros de registro del año 2016 completo (solo falta 2016-03-13, pero no es relevante). Para esto, se pueden mover todos otros ficheros de 2015 y de 2017 a otro directorio temporal aparte (fuera de `self.logs/`).

También será necesario modificar el script en Python `vectorize_secrepo.py` para aplicarlo a todas las IPs del año 2016 (no solo a las 10000 primeras, como en el caso anterior), haciendo los siguientes cambios:

- Comentar completamente las líneas: 21, 22 y 23, que en este caso no son aplicables, ya que las IPs indicadas en ellas solo aparecen en los registros de 2015, que no se utilizan ahora.
- Comentar la nueva línea 37, que contiene: "`if len(prevectors) >= 10000:`" y también la siguiente que contiene: "`continue`", para eliminar la limitación a las 10000 primeras IPs encontradas[6].

Una vez vectorizados los nuevos datos, se pueden ir haciendo pruebas con diferentes valores de parámetros en ambos algoritmos de *clustering*, e interpretar los diferentes resultados de agrupamientos para ver si es posible identificar algún grupo diferente, o si los elementos que no son agrupados (diferentes a los demás) pueden ser indicativos de actividad sospechosa. Hay que tener en cuenta que las direcciones IP que se usan como indicios en el libro (para identificar el posible cluster sospechoso) ya no están presentes en el nuevo conjunto de datos. Por lo tanto, es necesario utilizar otras formas de identificar cuáles pueden ser los clusters sospechosos.

Por ejemplo, con *K-Means* se puede ir probando con un número creciente de grupos partiendo desde el mínimo. Hay que observar si en las sucesivas distribuciones parece que siempre hay alguno de los grupos con un número de miembros (bajo) muy similar[7] y que empieza a tener un valor de *silhouette* alto. Después se puede observar si las IPs de ese grupo tienen algún patrón (ver lista ordenada), y verificar juntas las que tengan algo en común en los registros (reordenados), para comprobar si los *requests* también siguen algún patrón sospechoso.

También se puede intentar conseguir una clasificación con *DBSCAN* variando los hiperparámetros hasta que queden pocos elementos (IPs) sin clasificar (*noise points*[8]) y analizar la cohesión de los clusters (distancia media, *silhouette*) más pequeños, para comprobar si las IPs correspondientes (ordenadas) tienen algún patrón, o bien si los registros de *requests* (ordenados) de todas las IPs de ese grupo juntas[9] tienen algo en común o sospechoso, etc.

Como respuesta de este apartado (en el documento de la actividad explicado en el apartado 1.4), se deben indicar:

- Los casos y parámetros de clustering que se han probado inicialmente (de *K-Means* o *DBSCAN*) y porqué.
- Los resultados de estadísticas de las primeras pruebas que se han usado para decidir otros valores de los parámetros a probar y cómo se han usado para elegirlos.
- Si se han podido encontrar posibles clusters con IPs sospechosas de potenciales ataques, y qué características se observan en los resultados que se han obtenido en cada caso al analizar los registros correspondientes.

[6] Para poder quitar esta limitación, es necesario que la máquina en la que se ejecuta el código tenga disponible más de 1.6GiB de memoria RAM libre, debido al espacio necesario por `sklearn.metrics.pairwise_distances()`, en `stats_vectors.py`, para procesar las 20231 IPs diferentes en los registros de 2016.

[7] La etiquetas son arbitrarias (dependen de una inicialización aleatoria), y por tanto, no tienen por qué coincidir entre diferentes ejecuciones.

[8] En *DBSCAN* los puntos que no se han asignado a ningún grupo se etiquetan todos con "-1".

[9] Recuérdese que el comando "`grep`" permite buscar varias patrones o expresiones, colocadas en un fichero (una por línea), en múltiples ficheros simultáneamente, y que se puede inhibir el nombre del fichero del match en la salida (para poder ordenar por el contenido de líneas, etc.). También el filtro "`cut`" (estándar en Linux) permite seleccionar muy fácilmente un campo de todas las filas de una lista, etc. (o bien se puede modificar `label_notes.py` para que solo imprima las IPs (sin etiqueta delante) cuando solo se pide una etiqueta (con arg "`-l`").