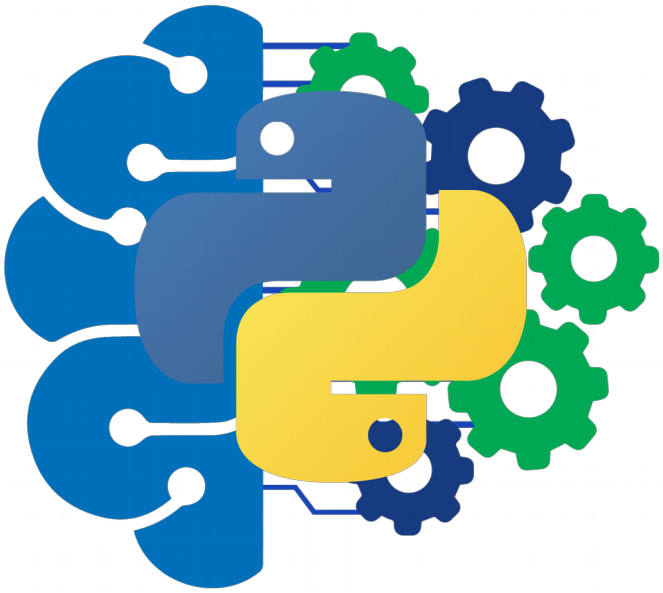


Trabajo Fin de Máster

APLICACIÓN DE TÉCNICAS DE MACHINE LEARNING A LA DETECCIÓN DE ATAQUES

MISTIC

Estudiante

José Manuel Rodríguez Rama

Consultor

Enric Hernández Jiménez

4 de Junio de 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Aplicación de técnicas de Machine Learning a la detección de ataques*
Nombre del autor:	*José Manuel Rodríguez Rama*
Nombre del consultor/a:	*Enric Hernández Jiménez*
Fecha de entrega (mm/aaaa):	06/2018
Titulación::	*MISTIC*
Área del Trabajo Final:	*Aplicación de técnicas de Machine Learning a la Seguridad*
Idioma del trabajo:	***Español***
Palabras clave	*Machine Learning Scikit-learn*

Resumen del Trabajo (máximo 250 palabras):

El presente proyecto forma parte del Trabajo Fin de Máster, concretamente del área “Aplicación de técnicas de Machine Learning a la Seguridad”, que había seleccionado como primera opción entre las distintas posibilidades, debido a mi interés por aprender acerca de ésta disciplina, cada vez más utilizada hoy en día.

El proyecto se desarrolla en distintas etapas, usando primero la plataforma Weka (entorno para análisis del conocimiento de la Universidad de Waikato) y un desarrollando posteriormente un script escrito en lenguaje Python que permitirá realizar el tratamiento del dataset y la posterior utilización de un modelo predictivo para la detección de conexiones maliciosas, concretamente utilizando la librería de software Scikit-Learn.

El dataset utilizado es “KDD Cup 1999” que incluye una amplia variedad de intrusiones de red simuladas en un entorno de red militar. Dicho dataset será analizado en el presente proyecto y se usará para entrenar, probar y ajustar el modelo seleccionado.

Se realizará una comparación de diversos algoritmos que se aplicarán al dataset, y se seleccionará el tipo que de mejores resultados prediciendo ataques, para posteriormente tratar de ajustarlo al máximo usando diferentes técnicas de Machine Learning e Ingeniería de datos.

Abstract (in English, 250 words or less):

This project is part of the Final Master Project, specifically the area of “Application of Machine Learning techniques to Security”, which I selected as the first option among different possibilities, due to my interest in learning about this subject, which is widely used nowadays.

The project is developed in different stages, first using the Weka platform (environment for knowledge analysis of the University of Waikato) and then developing a script written in Python that will do the pre-processing of the dataset, and the subsequent use of a predictive model for detecting malicious connections, specifically using the Scikit-Learn software library.

The dataset used is "KDD Cup 1999" which includes a wide variety of simulated network intrusions in a military network environment. This dataset will be analyzed in the present project and will be used to train, test and adjust the selected model.

A comparison of various algorithms will be made and applied to the dataset, and the algorithm with best results predicting attacks will be selected, and then try to find the best configuration using different techniques of Machine Learning and Data Engineering.

Dedicatoria

A mi mujer, por su paciencia y comprensión, por apoyarme durante todo este tiempo, y ayudarme a cumplir mis metas...

Índice

1. INTRODUCCIÓN

1.1. Justificación

La creciente demanda tecnológica experimentada en las últimas décadas ha favorecido el desarrollo de la disciplina científica del aprendizaje automático o aprendizaje de máquinas (del inglés “Machine Learning”), que es usado en ámbitos tan diversos como los motores de búsqueda, el diagnóstico médico, el análisis de mercado o la robótica.

El Machine Learning es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender, y generalizar comportamientos a partir de una información suministrada en forma de ejemplos.

Concretamente puede ser usada en seguridad informática para la detección de amenazas avanzadas, y presentan la ventaja de poder identificar ataques nunca antes vistos, dónde los sistemas tradicionales fallan al basar la detección únicamente en firmas o patrones conocidos.

1.2. Descripción del proyecto

El presente proyecto pretende demostrar la utilidad de Machine Learning, y su aplicación a la seguridad informática, concretamente a detección de conexiones maliciosas, para lo cual se realizará un análisis de un dataset que contiene un conjunto de conexiones clasificadas como normales o como un ataque determinado.

Posteriormente mediante herramientas de Machine Learning se estudiarán diferentes algoritmos y se seleccionará el mejor candidato, además de realizar diversas modificaciones y ajustes tanto del dataset como del algoritmo elegido, con el fin de conseguir la máxima precisión identificando ataques.

1.3. Objetivos del TFM

Los objetivos principales del proyecto son:

- Exponer la utilidad del aprendizaje automático, y particularmente su aplicación a la seguridad.
- Estudiar herramientas de aprendizaje automático.
- Comparar la precisión de diferentes algoritmos para un caso particular.
- Realizar un tratamiento de los datos con el fin de optimizar el proceso de aprendizaje.
- Optimizar los parámetros de entrenamiento y del modelo para conseguir la máxima precisión.
- Obtener un script que permita realizar todo el proceso automáticamente con los mejores resultados posibles.

1.4. Enfoque y método seguido

El proyecto se aborda de manera gradual, distinguiendo tres grandes fases que se exponen a continuación.

Formación e investigación:

Formación y estudio de posibilidades del sistema: esta fase, a pesar de no parecer la más compleja, es crítica, debido a que la formación sustenta el posterior desarrollo del sistema, y sin los conocimientos adecuados no se puede obtener un buen resultado.

Investigación y obtención de herramientas necesarias: en esta fase, y una vez se tiene una buena base teórica del proceso de aprendizaje automático, se dedica a investigar y seleccionar las herramientas necesarias para la realización del proyecto.

Desarrollo del sistema:

Una vez afianzados los conocimientos necesarios, y obtenidas las herramientas de Software y los sets de datos, se procede a la realización del sistema de detección como tal.

En un primer lugar se realiza un estudio detallado de los datos y se definen los atributos más importantes para la clasificación de las conexiones.

Tras haber realizado el estudio del dataset, se procede a probar diversos algoritmos para ver cuál es el que mejor resultado da, y se optimiza para obtener la máxima precisión.

Por último se diseña el script en lenguaje Python que automatizará el proceso.

Documentación:

Tras haber recopilado toda la información necesaria y diseñado el sistema se procede a documentar todo el proyecto mediante la realización de la memoria del TFM.

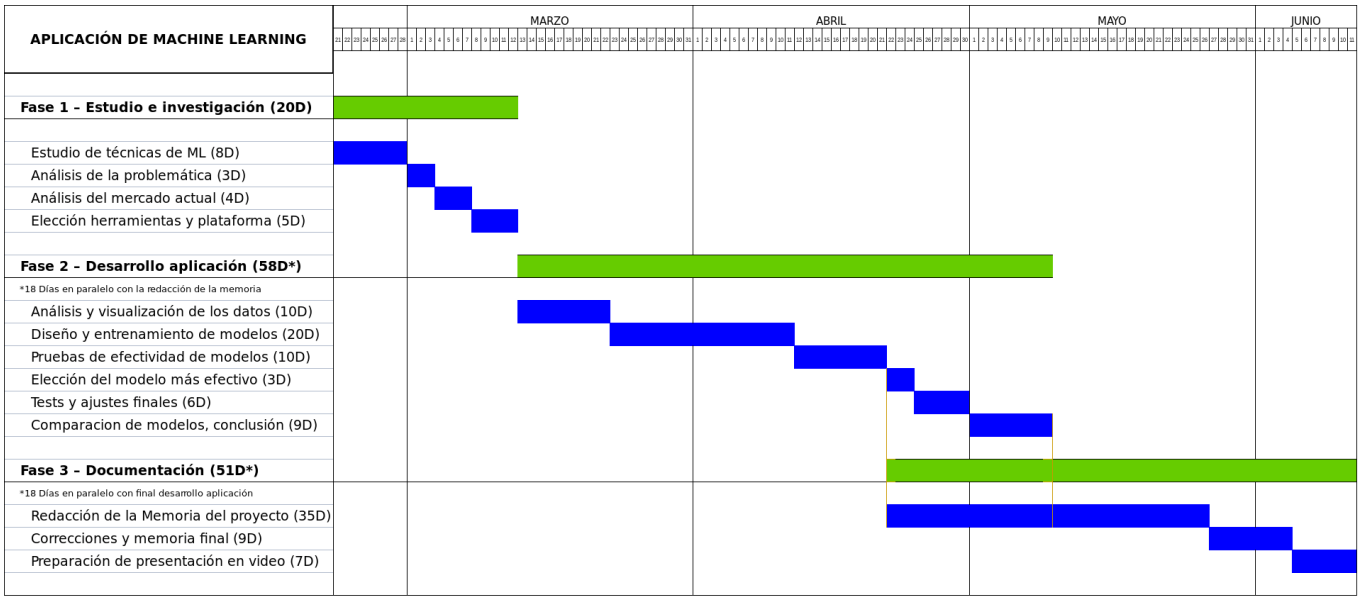
Tras la entrega de la memoria se realizará un vídeo en forma de presentación en diapositivas, mediante el cual se explicará todo el desarrollo del proyecto y la solución propuesta.

1.5. Planificación del Trabajo

Una vez establecidas las fases del proyecto se establece el tiempo necesario para realizar cada una. Se dispone de un total de 111 días para realizar el proyecto, y se dispondrá, aproximadamente una hora y media al día para realizarlo, es decir, unas 160 horas para la totalidad del trabajo distribuidas de la siguiente manera:

- Estudio e investigación: 20 días
- Desarrollo de la aplicación: 58 días (de los cuales 18 serán en paralelo con la memoria)
- Documentación: 51 días (de los cuales 18 serán en paralelo con el desarrollo de la aplicación)

Cronograma



1.6. Recursos empleados

Recursos de Hardware:

Para la realización del proyecto se utiliza un ordenador personal con procesador AMD Athlon(tm) X4 860K Quad Core con 3.7 GHz (4GHz en Turbo) y 2MB de memoria caché para una capacidad de procesamiento de 7386 Bogomips, un procesador de gama media que ofrece buenos rendimientos.

Se dispone de dos memorias de 8GB en DDR3, de 1600MHz de la marca Corsair modelo xms3 para un total de 16GB, suficientes para correr sin complicaciones el sistema operativo (Ubuntu Linux) y cargar el dataset completo en memoria.

Como disco duro se dispone de un SSD ADATA de 64GB conectado a la interfaz SATAIII de 6GB/s, con una velocidad de lectura secuencial de 550 MB/seg. Si bien la capacidad del disco es relativamente pequeña, es suficiente para nuestro caso y el hecho de usar un SSD frente a un HD supone una ventaja considerable en velocidad.

En cuanto a la tarjeta gráfica se dispone de una ATI Radeon HD 5450 de 400MHz y 1 GB de ram DDR3, y se trata una tarjeta de gama media-baja, pero suficiente para el presente proyecto, ya que Scikit no tiene soporte para GPU y sólo se necesitaría una buena tarjeta gráfica en caso de utilizar redes neuronales.

Además se dispone de dos pantallas distribuidas en forma de escritorio extendido, lo cual resulta muy útil para la visualización del dataset así como para realizar las correspondientes tareas de programación.

El coste total de todo el hardware no supera los 1000€, si bien no se ha realizado ningún gasto real en hardware al ser el mismo equipo se que usa de forma personal.

Recursos de Software:

En cuanto al Software instalado en el equipo, se destaca el sistema operativo Linux (Ubuntu 17.10 artful) con GNOME 3.26.1, con python 2.7 y java 1.8, sobre los cuales se han instalado las herramientas específicas para tratamiento de datos y Machine Learning que se exponen a continuación.

Scikit-Learn

Scikit-learn (anteriormente scikits.learn) es una biblioteca de aprendizaje de máquinas de software libre para el lenguaje de programación Python. Cuenta con varios algoritmos de clasificación, regresión y agrupación, incluyendo máquinas de vectores de soporte, bosques aleatorios, aumento de gradiente, k-means y DBSCAN, y está diseñado para interoperar con las bibliotecas numéricas y científicas de Python, NumPy y SciPy.

Pandas

Pandas es una biblioteca de software escrita como extensión de NumPy para manipulación y análisis de datos para el lenguaje de programación Python. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales. Es un software libre distribuido bajo la licencia BSD.

NumPy

NumPy es una extensión open source de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices.

Matplotlib

Matplotlib es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python y su extensión matemática NumPy. Proporciona una API, pylab, diseñada para recordar a la de MATLAB.

Weka

Weka (Waikato Environment for Knowledge Analysis, en español «entorno para análisis del conocimiento de la Universidad de Waikato») es una plataforma de software para el aprendizaje automático y la minería de datos escrito en Java y desarrollado en la Universidad de Waikato. Weka es software libre distribuido bajo la licencia GNU-GPL.

El paquete Weka contiene una colección de herramientas de visualización y algoritmos para análisis de datos y modelado predictivo, unidos a una interfaz gráfica de usuario para acceder fácilmente a sus funcionalidades.

Todo el software usado para el proyecto se puede obtener gratuitamente en internet por lo que no se ha realizado gasto alguno en términos de software.

1.7. Productos obtenidos

A continuación se describen brevemente los productos obtenidos tras la realización del presente trabajo.

Análisis del dataset

Se realiza un análisis detallado del dataset KDD99 Cup, se analiza su contenido y los atributos del mismo, a partir de los cuales se realizarán gráficos y rankings para decidir su importancia a la hora de entrenar modelos de Machine Learning.

Análisis de algoritmos

Se estudia el uso de distintos algoritmos para realizar predicciones usando el dataset KDD99 Cup, y se analizan los resultados en términos de rendimiento y confianza.

Aplicación (Script Python)

Se diseña un script que automatizará el proceso de carga del dataset, su posterior modificación y tests con el modelo elegido, así como un análisis y aplicación de mejoras a partir de una solución inicial, con el fin de obtener los mejores resultados.

1.8. Breve descripción de los otros capítulos de la memoria

Antecedentes

En este capítulo se describe la historia del aprendizaje automático desde sus inicios en los años 50 hasta la actualidad.

Estado del arte

Este capítulo se dedica a exponer los usos que se dan actualmente al Machine Learning, profundizando en sus aplicaciones a la seguridad informática.

Descripción

En este capítulo se realiza un estudio detallado del dataset, las posibles transformaciones a realizar sobre el mismo, y los algoritmos de ML que son más efectivos.

Aplicación

Detalla la creación de una aplicación, concretamente un script python, que automatiza el proceso de carga y transformación del dataset, y el uso de un modelo de predicción para identificar ataques.

Conclusiones

En este capítulo se exponen las conclusiones obtenidas tras la realización del proyecto, posibles mejoras, consecución de objetivos, y autoevaluación.

2. ANTECEDENTES

2.1. Historia del Machine Learning

Pese a lo que pueda parecer, la historia del Machine Learning se remonta a los años 50, concretamente en 1950 Alan Turing creaba el “Test de Turing” mediante el cual una máquina tendría que ser capaz de engañar a un humano y hacerlo pensar que era humana.

En los años posteriores se crearía el primer programa capaz de aprender jugando a las damas, y mejorando en cada partida, y en 1956, en la conferencia de Darthmouth, surgía por primera vez el término de “Inteligencia Artificial”. Un par de años más tarde Frank Rosenblatt diseñaría el Perceptrón, la primera red neuronal artificial, y en 1967 se escribiría el algoritmo “Nearest Neighbor” considerado como el nacimiento al campo del reconocimiento de patrones.

En la segunda mitad de la década de los 70 el campo sufrió su primer “Invierno” debido a recortes en investigación tras numerosos años de expectativas y pocos avances, sin embargo, en los años 80 nacieron los sistemas expertos, basados en reglas, que fueron rápidamente adoptados en el sector corporativo, como sería caso del “Explanation Based Learning” (EBL), donde un computador analiza datos de entrenamiento y crea reglas generales que le permiten descartar los datos menos importantes.

A finales de los 80, y durante la primera mitad de los 90, hubo una segunda decaída, y el ML no se recuperaría hasta entrados los 2000, tras el giro desde el enfoque orientado al conocimiento (knowledge-driven), hasta uno orientado a los datos (data-driven).

Gracias al aumento de la capacidad de procesamiento de los sistemas y a la gran cantidad de datos disponibles, las empresas pudieron transformar sus negocios hacia el dato e incorporar técnicas de Machine Learning en sus procesos, productos y servicios para obtener ventajas sobre la competencia, lo que ha llevado a que actualmente se esté experimentando una tercera explosión en el sector, con nuevas ideas y aplicaciones cada día.

2.2. Acontecimientos destacados

1997 — El ordenador Deep Blue, de IBM vence al campeón mundial de ajedrez Gary Kaspárov.

2006 — Geoffrey Hinton acuña el término “Deep Learning” (Aprendizaje Profundo) para explicar nuevas arquitecturas de Redes Neuronales profundas.

2011 — El ordenador Watson de IBM vence a sus competidores humanos en el concurso Jeopardy que consiste en contestar preguntas formuladas en lenguaje natural.

2012 — Jeff Dean, de Google, con la ayuda de Andrew Ng (Universidad de Stanford), lideran el proyecto GoogleBrain, que desarrolla una Red Neuronal Profunda utilizando toda la capacidad de la infraestructura de Google para detectar patrones en vídeos e imágenes.

2012 — Geoffrey Hinton lidera el equipo ganador del concurso de Visión por Computador a Imagenet utilizando una Red Neuronal Profunda (RNP). El equipo venció por un amplio margen de diferencia, dando nacimiento a la actual explosión de Machine Learning basado en RNPs.

2012 — El laboratorio de investigación Google X utiliza GoogleBrain para analizar autónomamente vídeos de Youtube y detectar aquellos que contienen gatos.

2014 — Facebook desarrolla DeepFace, un algoritmo basado en RNPs que es capaz de reconocer a personas con la misma precisión que un ser humano.

2014 — Google compra DeepMind, una startup inglesa de Deep Learning que recientemente había demostrado las capacidades de las Redes Neuronales Profundas con un algoritmo capaz de jugar a juegos de Atari simplemente viendo los píxeles de la pantalla, y batir a humanos expertos en algunos de esos juegos.

2015 — Microsoft crea el “Distributed Machine Learning Toolkit”, que permite la distribución eficiente de problemas de machine learning en múltiples computadores.

2015 — Elon Musk y Sam Altman, entre otros, fundan la organización sin ánimo de lucro OpenAI, dotándola de 1000 Millones de dólares con el objetivo de asegurar que el desarrollo de la Inteligencia Artificial tenga un impacto positivo en la humanidad.

2016 – Google DeepMind vence en el juego Go (considerado uno de los juegos de mesa más complicados) al jugador profesional Lee Sedol por 5 partidas a 1. Jugadores expertos de Go afirman que el algoritmo fue capaz de realizar movimientos “creativos” que no se habían visto hasta el momento.

3. ESTADO DEL ARTE

3.1. Estado actual del Machine Learning

Actualmente es posible hallar aplicaciones de Machine Learning en ámbitos muy diversos, como el asistente en un teléfono móvil, un antivirus, un análisis de mercado, o el un control de accesos a un edificio. Se estima que a finales de 2018 un 62% de las empresas habrán utilizado algún tipo de tecnología de inteligencia artificial, lo que la hace una de las tecnologías más importantes hoy día.

A continuación describimos algunas de las principales aplicaciones de Machine Learning a día de hoy:

1.- Predicciones: Se almacena información de manera digital, que es explotada posteriormente para la toma de decisiones en las empresas, lo que se conoce como Business Intelligence. Mediante el uso de algoritmos los datos se convierten en predicciones que ayudan a las empresas a sabér qué se va a vender más, y predecir ganancias o pérdidas en el futuro.

2.- Detección de intrusos: Los sistemas de IA aprenden el comportamiento cotidiano de los usuarios autorizados, entonces cuando el comportamiento de un usuario cambia, el sistema decide si se trata de una amenaza y genera los bloqueos necesarios. Estos sistemas son también conocidos como “detectores de anomalías”.

3.- Antivirus: Segun Kaspersky se crean 325,000 nuevos malwares diarios, existen tantos virus que analizar cada virus en cada archivo es un problema intratable, por ello en la actualidad los antivirus contienen un algoritmo que aprende características esenciales de los malwares actualmente conocidos, reduciendo la complejidad de detectarlos.

4.- Clasificación de texto: Se trata de algoritmos capaces de clasificar texto en base a su contenido, por ejemplo, clasificar páginas en base al nivel del sentimiento que provocan. También se pueden analizar comentarios de redes sociales con fines de marketing.

5.- Productividad: Otra aplicación del ML en empresas es la mejora de la productividad; mediante algoritmos se entrenan para comprender el flujo de trabajo de la empresa y encontrar puntos cruciales que pueden ser mejorados, como las líneas de producción o los costos de los productos ofrecidos.

6.- Recomendación de productos: Actualmente, varios de estos sistemas incorporan algoritmos de aprendizaje, que utilizan el historial de compras y búsqueda de productos, y sugieren a los clientes productos nuevos. Otra aplicación similar muy usada es la publicidad dirigida en internet.

7.- Bots de soporte: Se trata de sistemas que simulan ser una persona y pueden ser entrenados para ofrecer un primer apoyo a los clientes, y cuando ya no son capaces de resolver preguntas muy complejas, ceden el trabajo a un humano.

8.- Customer churn: Otra de las aplicaciones en el mundo empresarial es el customer churn, o pérdida de clientes. Mediante el uso de sistemas de aprendizaje, y usando toda la información relacionada con el cliente, se identifica la razón por la que un cliente se ha ido, e incluso se puede proponer una estrategia para recuperarlo.

9.- Lingüística computacional: Este campo mezcla lingüística, informática e inteligencia artificial, pues se encarga de perfeccionar las interacciones entre las máquinas y los lenguajes humanos. Sus aplicaciones más conocidas son la traducción automática de idiomas, sistemas de recuperación de información y producción automática de textos.

10.- Deep learning: Es una de las áreas de mayor crecimiento, utiliza algoritmos multinivel para crear diferentes niveles de abstracción de la información. Algunas aplicaciones son el reconocimiento de imágenes, voz, predicción de tendencias digitales y perfeccionamiento de procesos.

11.- Biométrica: Se ocupa de la identificación, medición y análisis de aspectos físicos del cuerpo y su forma. Es una de las tecnologías más tangibles dentro de la inteligencia artificial y tiene distintas aplicaciones como el diagnóstico médico o la seguridad para el acceso a información en dispositivos móviles

3.2. Aplicaciones de Machine Learning a la seguridad informática.

El aprendizaje de máquinas es usado en seguridad en diversos ámbitos, como pueden ser la identificación de comportamientos o entidades maliciosas, ya sea hackers, malware, o simplemente comportamientos no deseados de un usuario.

Los sistemas tradicionales, basados en firmas, se emplean especialmente para detectar ataques conocidos, como por ejemplo un virus que contiene un determinado fichero o un ataque de SQL Injection que contiene una determinada cadena de texto conocida, sin embargo los sistemas de Machine Learning basados en detección de anomalías identifican patrones de comportamiento que representen una desviación de la “normalidad”, siendo más potentes que los basados en firmas al permitir detectar ataques desconocidos por el sistema.

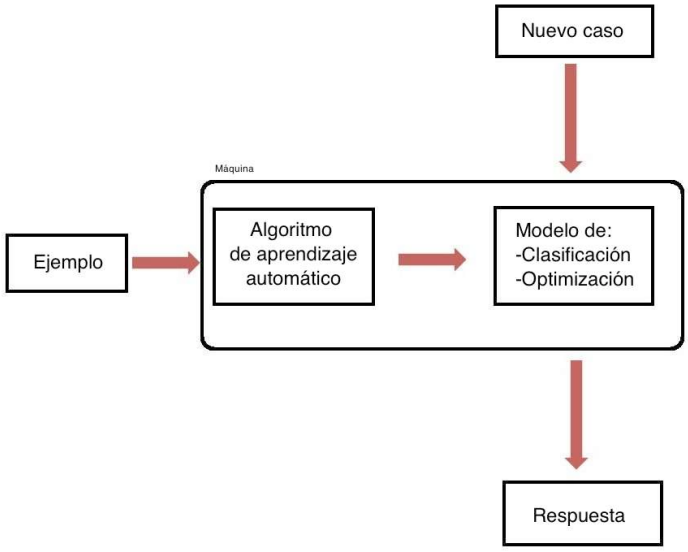
Además los sistemas basados en firmas requieren que éstas sean actualizadas constantemente, y por supuesto no pueden detectar ataques no conocidos (zero-day attacks). Otra importante ventaja es que la “normalidad” se define para cada sistema, dificultando a los atacantes la preparación de herramientas que pasen desapercibidas por el IDS.

Existen también sistemas híbridos que combinan la detección basada en firmas con la detección basada en anomalías. En la práctica apenas existen sistemas basados en detección de anomalías puros.

Para encontrar anomalías se debe primero definir qué es normal, para lo cual no se puede utilizar simplemente la estadística, ya que se pueden dar muchas anomalías a nivel estadístico que rara vez representan un ataque, como por ejemplo un aumento en el tráfico de la red.

Para resolver este problema se utilizan algoritmos que aprendan a distinguir las anomalías ya sea a partir de ejemplos ya etiquetados (aprendizaje supervisado) o buscando patrones en entradas sin etiquetar que permitan indentificar posteriormente nuevas entradas (aprendizaje no supervisado).

A continuación podemos ver una figura que explica el funcionamiento básico del proceso de Machine Learning

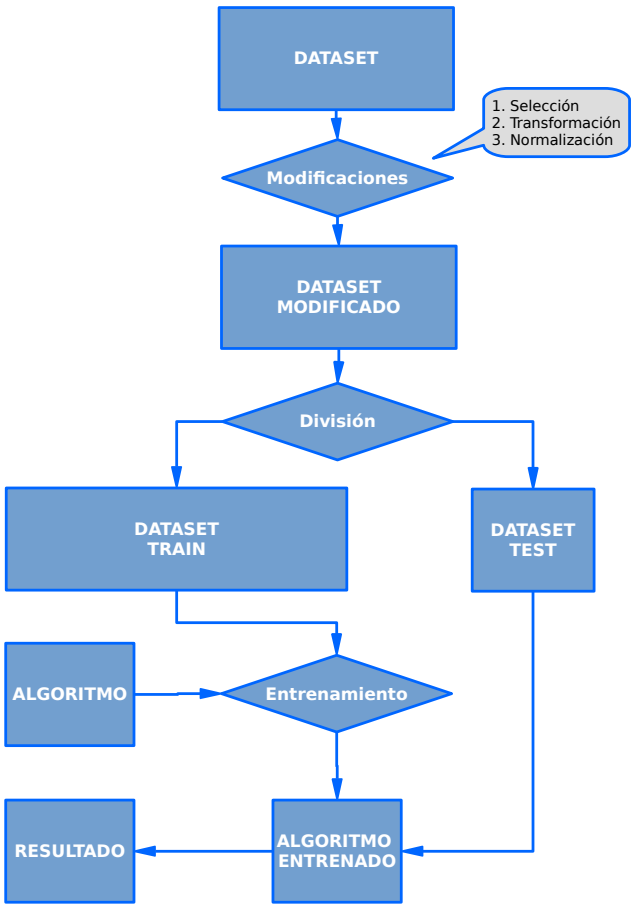


4. DESCRIPCIÓN

En el presente capítulo expondremos la solución propuesta. Se pretende crear una imagen detallada de la misma, presentando los diversos elementos que componen el sistema y su interacción.

4.1. Diagrama de bloques

Como se puede ver en el siguiente diagrama de bloques, se parte del set de datos, y se realizan las modificaciones necesarias para aumentar la precisión y el rendimiento, dividiéndolo posteriormente en dos partes, una para entrenar el algoritmo propuesto, y la otra para realizar tests con el mismo una vez entrenado y comprobar su precisión prediciendo ataques.



4.2. Conjunto de datos (dataset)

Un conjunto de datos (conocido también por el anglicismo: dataset) es una colección de datos habitualmente tabulada.

En general un conjunto de datos corresponde a los contenidos de una única tabla de base de datos o una única matriz de datos estadística, donde cada columna de la tabla representa una variable en particular, y cada fila representa a un miembro determinado del conjunto de datos en cuestión.

a) Descripción del dataset

En el presente proyecto se utilizará el dataset KDDCUP'99, utilizado ampliamente para la evaluación de sistemas de detección de anomalías. El dataset se obtiene del programa de evaluación de IDS DARPA'98, que consiste en una captura durante 7 semanas de tráfico de red, concretamente unos 5 millones de conexiones capturadas en 4GB de datos tcpdump.

El dataset de KDD consiste en aproximadamente 4.900.000 de vectores de conexiones, cada uno con 41 atributos, y etiquetados como "normal" o como un ataque determinado.

b) Descripción de los ataques

El dataset contiene un total de 22 ataques, agrupados en 4 grandes categorías:

1.- DoS (Denegación de Servicio): El atacante intenta prevenir a usuarios auténticos que puedan usar un servicio, ya sea mediante una sobrecarga de los recursos, o un desbordamiento de buffer. Las etiquetas relativas a estos ataques son: *Neptune, Smurf, Pod, Teardrop, Land.*

2.- Probing: Los ataques de Probing o Surveillance, tienen la finalidad de obtener información acerca de la configuracion de una computadora o una red, que le permita posteriormente atacar el equipo. Las etiquetas relativas a estos ataques son: *ipsweep, nmap, satan, portsweep.*

3.- R2L: El ataque Remote-to-local consiste en el envío de paquetes de forma remota de un intruso a una maquina local en una red, sin que este tenga una cuenta en esa máquina, para mediante alguna vulnerabilidad ganar acceso local. Las etiquetas relativas a estos ataques son: *warezclient, guess_passwd, warezmaster, ftp_write, imap, multihop, phf, spy.*

4.- U2R: El ataque User-to-root, es un intento de un usuario no autorizado de ganar privilegios de administrador. El atacante empieza con un acceso normal a una cuenta de usuario del sistema y trata de explotar una vulnerabilidad para ganar acceso root. Las etiquetas relativas a estos ataques son: *perl, buffer_overflow, rootkit, loadmodule.*

c) Descripción de los atributos

Llamamos atributos a aquellas características que describen cada una de las instancias del conjunto de datos. Las denominaciones se usan indistintamente en función del autor y del contexto. Por ejemplo en el caso de un coche hablaríamos de su marca, su color, su cilindrada, etc. Si visualizáramos los datos en una hoja de cálculo, los atributos serían las columnas.

En el dataset KDD99 existen un total de 41 atributos que están divididos en cuatro grupos que exponemos a continuación:

Atributos básicos: Pueden ser obtenidos de los encabezados de los paquetes sin inspeccionar el payload.

Atributos de contenido: El dominio del conocimiento es usado para acceder al payload de los paquetes TCP originales, que incluyen características como el número de intentos de autenticación fallidos

Atributos de Trafico basados en tiempo: Estos atributos capturan propiedades que maduran en una ventana temporal de 2 segundos. Un ejemplo es el número de conexiones al mismo host durante 2 segundos.

Atributos de Tráfico basados en el Host: Utiliza una ventana histórica estimada sobre el número de conexiones, en este caso 100, en vez del tiempo, para poder detectar ataques cuando el intervalo es mayor a 2 segundos.

En el anexo 1 se define cada uno de los atributos, y se proporciona un gráfico asociado a cada uno para visualizar su peso con respecto a los ataques.

d) Ingeniería de Atributos

La ingeniería de atributos es el proceso por el cual se realiza un análisis, limpieza, estructuración, transformación y escalado de los atributos del dataset, con el fin de optimizar la información que el modelo de predicción va a recibir, y poder aumentar la confianza y el rendimiento.

En el presente trabajo se realizará, en primer lugar, una selección de atributos combinando las técnicas de correlación y ganancia de la información, para eliminar los atributos menos interesantes para el modelo.

Posteriormente se transforman los atributos categóricos en numéricos para poder usarlos con los modelos que no acepten atributos categóricos. Una vez hecho esto se realiza un escalado de atributos para que tome valores entre 0 y 1 y mejorar el desempeño del algoritmo.

4.2.1 Selección de atributos

Mediante la selección de atributos (Feature Selection, en inglés) conseguimos limpiar la entrada de datos que vamos a pasar al algoritmo, mejorando su comportamiento. Eliminar el ruido en los datos no sólo favorece el rendimiento a la hora de entrenar el algoritmo, sino que puede incluso ser positivo para aumentar la confianza, y obtener mejores resultados que si se usara el dataset completo.

Se propone como objetivo eliminar aproximadamente la cuarta parte de los atributos y evaluar los cambios en rendimiento y confianza, para lo cual, mediante la aplicación escrita en python, se calculará el tiempo dedicado a entrenar el modelo, así como la precisión detectando ataques.

Las razones para usar Selección de atributos son: reducir el tiempo dedicado a entrenar el proceso, reducir la complejidad del modelo y hacerlo más sencillo de interpretar, aumentar la precisión (si se elige el subset correcto), y reducir el sobre-entrenamiento.

Para decidir qué atributos vamos a seleccionar se usa el Framework WEKA, en el cual cargamos el dataset (en este caso el dataset 10% para agilizar el proceso), y usamos la herramienta de ranking para selección de atributos, para dos los métodos más comunes, la selección basada en correlación y la basada en ganancia de la información.

Selección de atributos basada en correlación: En este tipo de selección se evalúan los subconjuntos de atributos partiendo de la base de que los subconjuntos de características buenas contienen características altamente correlacionadas con la clasificación, pero no están correlacionadas entre sí.

Resultados de la evaluación en WEKA:

Evaluator: weka.attributeSelection.CorrelationAttributeEval
Search: weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation: kdd_cup_1999
Instances: 494020
Attributes: 42
Evaluation mode: evaluate on all training data

Ranking:

0.81039 24 srv_count
0.77746 36 dst_host_same_src_port_rate
0.7612 23 count
0.75803 2 protocol_type
0.65875 3 service
0.61224 29 same_srv_rate
0.61198 33 dst_host_srv_count
0.61054 34 dst_host_same_srv_rate
0.57305 4 flag
0.53779 38 dst_host_serror_rate
0.53763 25 serror_rate
0.53708 39 dst_host_srv_serror_rate
0.53705 26 srv_serror_rate
0.47752 12 logged_in
0.39341 32 dst_host_count
0.24638 35 dst_host_diff_srv_rate
0.23238 30 diff_srv_rate
0.23126 40 dst_host_rerror_rate
0.23062 41 dst_host_srv_rerror_rate
0.22935 28 srv_rerror_rate
0.22918 27 rerror_rate
0.22865 31 srv_diff_host_rate
0.16406 37 dst_host_srv_diff_host_rate
0.07558 1 duration
0.04331 8 wrong_fragment
0.0368 10 hot
0.03571 22 is_guest_login
0.03189 19 lnum_access_files
0.02769 6 dst_bytes
0.0123 17 lnum_file_creations
0.01056 18 lnum_shells
0.00929 14 lroot_shell
0.00779 11 num_failed_logins
0.0065 16 lnum_root
0.00566 7 land
0.00551 13 lnum_compromised
0.00532 15 lsu_attempted
0.00227 9 urgent
0.00202 5 src_bytes
0 20 lnum_outbound_cmds
0 21 is_host_login

Selección de atributos basada en ganancia de información: En este tipo de selección determina el decremento de entropía al conocer el resultado de un suceso.

Resultados de la evaluación en WEKA:

Evaluator: weka.attributeSelection.InfoGainAttributeEval
Search: weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation: kdd_cup_1999
Instances: 494020
Attributes: 42

Ranking:

1.4384029 5 src_bytes
1.3902034 23 count
1.3552348 3 service
1.0977182 24 srv_count
1.0961848 36 dst_host_same_src_port_rate
1.0159265 2 protocol_type
0.9054253 33 dst_host_srv_count
0.8997895 35 dst_host_diff_srv_rate
0.8713826 34 dst_host_same_srv_rate
0.7844647 30 diff_srv_rate
0.7773695 29 same_srv_rate
0.7645689 4 flag
0.5820167 6 dst_bytes
0.5662843 38 dst_host_serror_rate
0.5449353 25 serror_rate
0.5438835 39 dst_host_srv_serror_rate
0.521429 26 srv_serror_rate
0.4364079 12 logged_in
0.3470899 32 dst_host_count
0.3060166 37 dst_host_srv_diff_host_rate
0.183226 31 srv_diff_host_rate
0.1722753 40 dst_host_rerror_rate
0.1619854 41 dst_host_srv_rerror_rate
0.1223836 27 rerror_rate
0.1111194 28 srv_rerror_rate
0.0631425 1 duration
0.0502661 10 hot
0.039662 13 lnum_compromised
0.0269304 8 wrong_fragment
0.0061675 22 is_guest_login
0.0029189 16 lnum_root
0.0022768 19 lnum_access_files
0.00167 17 lnum_file_creations
0.0015435 11 num_failed_logins
0.0008922 14 lroot_shell
0.0006715 7 land
0.0004195 18 lnum_shells
0.0000961 9 urgent
0.0000793 15 lsu_attempted
0 20 lnum_outbound_cmds
0 21 is_host_login

Una vez obtenidos los dos rankings, seleccionamos los 12 atributos con menor puntuación en cada caso, y buscamos los atributos comunes en ambas listas (10), que eliminaremos de nuestro dataset.

Ranking: Correlación (atributos comunes en verde)

0.0061675	22 is_guest_login
0.0029189	16 lnum_root
0.0022768	19 lnum_access_files
0.00167	17 lnum_file_creations
0.0015435	11 num_failed_logins
0.0008922	14 lroot_shell
0.0006715	7 land
0.0004195	18 lnum_shells
0.0000961	9 urgent
0.0000793	15 lsu_attempted
0	20 lnum_outbound_cmds
0	21 is_host_login

Ranking: Ganancia de información (atributos comunes en verde)

0.0123	17 lnum_file_creations
0.01056	18 lnum_shells
0.00929	14 lroot_shell
0.00779	11 num_failed_logins
0.0065	16 lnum_root
0.00566	7 land
0.00551	13 lnum_compromised
0.00532	15 lsu_attempted
0.00227	9 urgent
0.00202	5 src_bytes
0	20 lnum_outbound_cmds
0	21 is_host_login

4.2.2 Codificación de atributos

La codificación de atributos consiste en convertir, mediante una función, los atributos categóricos en numéricos, ya que dependiendo del algoritmo este podría no soportar los atributos categóricos.

Existen varios métodos, desde realizar la transformación manualmente, hasta utilizar herramientas que automatizan el proceso, como las contenidas en las librerías de Python como Pandas y Scikit.

Existen tres principales técnicas para convertir los atributos:

Buscar y Reemplazar, en casos en los que queramos por ejemplo reemplazar todos los atributos de valor “dos” por 2. Requiere realizar una parte del trabajo manualmente.

Codificado de etiquetas, que convierte simplemente cada valor en una columna por un número que tenga relación con su contenido. Es la más sencilla de las tres.

Codificado “One Hot”, que agrega columnas dependiendo del número de valores a codificar y da valores 1 o 0 (True o False), de forma que la codificación no aporta peso a un valor en particular. Tiene la desventaja de aumentar el número de columnas del dataset.

En nuestro caso los atributos a convertir serían *protocol_type, service, y flag*. Para codificar estos valores vamos a usar Scikit-learn y la función LabelEncoder incluída en sklearn.preprocessing, por ser una opción sencilla que no aumentará el número de columnas.

4.2.3 Escalado de los atributos

El escalado de los atributos, también conocido como normalizado de los datos, es otro de los pasos que se realizan durante el proceso de preprocesado del dataset.

Debido a que los datos sin procesar pueden variar ampliamente, algunos algoritmos no funcionan correctamente sin realizar la normalización, debido a que la mayoría de los clasificadores calculan la distancia Euclidea entre dos puntos, y si uno de los atributos tiene un rango muy grande de valores, la distancia estará gobernada por este atributo en particular.

Por esta razon los atributos deben ser normalizados, de forma que cada uno contribuya de manera proporcional a la distancia final. Destacamos dos tipos de escalado, el escalado estándar (Standard Scaler) y el escalado de variables (Feature Scaling o MinMax Scaler).

En el **escalado de variables** se comprimen los datos de etnrada entre unos límites empíricos (el máximo y el mínimo de la variable), por lo que si existe ruído en la entrada éste será ampliado.

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

En el **escalado estándar**, a cada dato se le resta la media de la variable, y se divide por la desviación típica. En este caso hay que tener en cuenta que los dos estadísticos que se usan (media y desviación típica) son muy sensibles a valores anómalos (muy grandes o muy pequeños con respecto al resto).

$$X_{normalized} = \frac{X - X_{mean}}{X_{stddev}}$$

En este trabajo proponemos el uso del escalado de variables, ya que es el que mejor se ajusta a los valores de los atributos que tenemos. Para normalizar estos valores vamos a usar Scikit-learn y la función MinMaxScaler incluída en sklearn.preprocessing.

4.3. Algoritmos

Un algoritmo es una secuencia de instrucciones que se llevan a cabo para representar un modelo de solución para un determinado tipo de problemas.

En Machine Learning los algoritmos tienen diversas funciones dependiendo de su tipo. Se clasifican en tres tipos dependiendo si se trata de aprendizaje automático supervisado, no supervisado, o de refuerzo.

Los algoritmos de aprendizaje supervisado realizan sus predicciones basándose en un conjunto de ejemplos etiquetados. El algoritmo buscará los patrones en esas etiquetas y una vez que encuentra el mejor patrón posible, puede ser usado para hacer prediciones de datos de prueba sin etiquetar.

Distinguimos distintos tipos de algoritmos en el aprendizaje supervisado:

Clasificación, cuando los datos se usan para predecir una categoría, multiclase si se clasifica en varias clases, o binomial si sólo se clasifica en dos (por ejemplo bueno o malo).

Regresión, cuando se predice un valor (como por ejemplo el precio de unas acciones).

Detección de anomalías, cuando se busca identificar datos que no son habituales, es decir, que difieren de unos patrones identificados como actividad normal.

Los algoritmos de aprendizaje no supervisado son aquellos que no tienen etiquetas asociadas a ellos, por lo que su objetivo es organizar los datos de alguna manera o describir su estructura. Puede realizarse una agrupación en clústeres o buscar diferentes maneras de examinar datos complejos para hacerlos más simples y organizarlos.

Los algoritmos de aprendizaje de refuerzo eligen una acción en respuesta a cada punto de datos y reciben una señal de recompensa más adelante para indicar cómo de buena fue la decisión, que es usada por el algoritmo para modificar la estrategia y obtener la mayor recompensa.

4.3.1. Selección del algoritmo

Dado que nuestro dataset contiene instancias de conexiones etiquetadas como normal o ataque, tenemos que abordar el problema desde el punto de vista de los algoritmos de aprendizaje supervisado, concretamente usando algoritmos de clasificación, ya sea binaria, separando todas las conexiones normales de los que no lo sean, o multiclase, identificando cada ataque por separado.

Para seleccionar el algoritmo que ofrezca los mejores resultados en nuestro caso, usaremos Weka con el dataset KD99 (10%) y abriremos el Weka Explorer, en la pestaña Classify, dónde correremos diversos algoritmos sobre nuestro dataset y compararemos los resultados obtenidos. El dataset se separa en dos partes, un 85% para entrenar y el 15% restante para test y se usan los parámetros por defecto para todos los algoritmos.

A continuación mostramos los resultados obtenidos para distintos algoritmos, tomaremos como referencia para medir el rendimiento el tiempo de creación del modelo, y para la confianza el porcentaje de instancias clasificadas correctamente:

Random Tree:

Time taken to build model:		**7.71 seconds**
Time taken to test model on test split:		0.23 seconds
Correctly Classified Instances	74049	**99.9271 %**
Incorrectly Classified Instances	54	0.0729 %
Kappa statistic		0.9988
Mean absolute error		0.0001
Root mean squared error		0.0079
Relative absolute error		0.1228 %
Root relative squared error		4.9257 %
Total Number of Instances		74103

Random Forest:

Time taken to build model:		**387.39 seconds**
Time taken to test model on test split:		3.51 seconds
Correctly Classified Instances	74074	**99.9609 %**
Incorrectly Classified Instances	29	0.0391 %
Kappa statistic		0.9993
Mean absolute error		0.0001
Root mean squared error		0.005
Relative absolute error		0.1326 %
Root relative squared error		3.1189 %
Total Number of Instances		74103

J48:

Time taken to build model:		**64.68 seconds**
Time taken to test model on test split:		0.55 seconds
Correctly Classified Instances	74069	**99.9541 %**
Incorrectly Classified Instances	34	0.0459 %
Kappa statistic		0.9992
Mean absolute error		0.0001
Root mean squared error		0.0062
Relative absolute error		0.1053 %
Root relative squared error		3.8962 %
Total Number of Instances		74103

Naïve Bayes:

Time taken to build model:		**5.1 seconds**
Time taken to test model on test split:		27.44 seconds
Correctly Classified Instances	68928	**93.0165 %**
Incorrectly Classified Instances	5175	6.9835 %
Kappa statistic		0.8845
Mean absolute error		0.0061
Root mean squared error		0.0758
Relative absolute error		11.7758 %
Root relative squared error		47.2705 %
Total Number of Instances		74103

SVM:

Time taken to build model:		**486.68 seconds**
Time taken to test model on test split:		8.72 seconds
Correctly Classified Instances	74033	**99.9055 %**
Incorrectly Classified Instances	70	0.0945 %
Kappa statistic		0.9984
Mean absolute error		0.0794
Root mean squared error		0.1961
Relative absolute error		154.4792 %
Root relative squared error		122.3498 %
Total Number of Instances		74103

Regresión Logística:

Time taken to build model:		**6016.51 seconds**
Time taken to test model on test split:		1.94 seconds
Correctly Classified Instances	74022	**99.8907 %**
Incorrectly Classified Instances	81	0.1093 %
Kappa statistic		0.9982
Mean absolute error		0.0002
Root mean squared error		0.0094
Relative absolute error		0.3269 %
Root relative squared error		5.8756 %
Total Number of Instances		74103

Podemos observar que todos los algoritmos dan resultados alrededor del 99% de precisión excepto Naïve Bayes con el que obtenemos solamente un 93%, siendo los árboles los que mejores resultados generales dan, y en rendimiento Random Tree, con una considerable diferencia, el que más rápido se ejecuta. Con Regresión logística se obtiene la mejor precisión, pero el tiempo para crear el modelo es demasiado elevado. Se seleccionan por lo tanto como tipo los árboles de decisión para nuestra aplicación. El objetivo será mejorar los valores de Weka en cuanto a precisión y rendimiento se refiere.

5. APLICACIÓN

A partir de los resultados obtenidos en los apartados anteriores, se diseña una aplicación que permite cargar el dataset KD99 completo, realizar las modificaciones sobre el , y después utilizarlo para entrenar un arbol de decisión y medir su precisión detectando conexiones maliciosas.

Para escribir la aplicación se utiliza el lenguaje de scripting Python y la librería de Machine Learning Scikit-Learn, así como otras librerías para tratamiento y computación de datos como Pandas, Numpy y la herramienta de gráficos Matplotlib. El script está escrito usando jupyter notebook, una aplicación que permite ordenar y documentar el código, así como realizar ejecuciones por partes del mismo.

5.1. Código

Importación de módulos:

A continuación se muestran todos los módulos que se deben importar para correr la aplicación correctamente

In [1]:
```
import pandas as pd #Dataframe, Series
import numpy as np #Paquetes de Scientific computing, Arrays
from matplotlib import pyplot as plt #Graficos

from sklearn.model_selection import train_test_split #Dividir Dataset en train y test
from sklearn.preprocessing import LabelEncoder #Pasar datos categoricos a numericos
from sklearn import preprocessing #Normalizacion de datos

#Imports dibujo arbol de decision
import graphviz
import StringIO as io
import pydotplus
import imageio

import time #Medir tiempo de entrenamiento

%matplotlib inline
```

Tenemos los módulos de pandas, numpy y matplotlib, así como los correspondientes a Scikit-learn, y unos auxiliares para dibujar el árbol de decisión (graphviz, StringIO, etc.) y medir el tiempo de entrenamiento de los modelos (time).

Carga del dataset:

En esta parte de nuestro script cargamos el dataset KDD99 completo con Pandas, e indicamos el nombre de las columnas (atributos).

In [2]:
```
col_names = ["duration","protocol_type","service","flag","src_bytes",
    "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
    "logged_in","num_compromised","root_shell","su_attempted","num_root",
    "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
    "is_host_login","is_guest_login","count","srv_count","serror_rate",
    "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
    "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
    "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
    "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
    "dst_host_rerror_rate","dst_host_srv_rerror_rate","label"]

data = pd.read_csv('../dataset/kddcup.data.corrected', header=None, names = col_names)
```

Visualización de los datos

Una vez cargado el dataset, el script nos permite realizar una visualización rápida de los datos cargados y de sus características con las funciones describe() que realiza un conteo de los datos numéricos, head() que muestra los datos de las primeras filas, e info() que devuelve una lista de columnas y nos indica de qué tipo son los datos que contiene.

In [24]:
```
data.describe()
```

Out[24]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromis
count	4.898431e+06	4.898431e+06	4.898431e+06	4.898431e+06	4.898431e+06	4.898431e+06	4.898431e+06	4.898431e+06	4.898431e+06	4.898431e+
mean	4.834243e+01	1.834621e+03	1.093623e+03	5.716116e-06	6.487792e-04	7.961733e-06	1.243766e-02	3.205108e-05	1.435290e-01	8.088304e
std	7.233298e+02	9.414311e+05	6.450123e+05	2.390833e-03	4.285434e-02	7.215084e-03	4.689782e-01	7.299408e-03	3.506116e-01	3.856481e+
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+
25%	0.000000e+00	4.500000e+01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+
50%	0.000000e+00	5.200000e+02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+
75%	0.000000e+00	1.032000e+03	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+
max	5.832900e+04	1.379964e+09	1.309937e+09	1.000000e+00	3.000000e+00	1.400000e+01	7.700000e+01	5.000000e+00	1.000000e+00	7.479000e+

8 rows × 38 columns

In [25]:
```
data.head()
```

Out[25]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_dif
0	0	tcp	http	SF	215	45076	0	0	0	0	...	0	0.0	
1	0	tcp	http	SF	162	4528	0	0	0	0	...	1	1.0	
2	0	tcp	http	SF	236	1228	0	0	0	0	...	2	1.0	
3	0	tcp	http	SF	233	2032	0	0	0	0	...	3	1.0	
4	0	tcp	http	SF	239	486	0	0	0	0	...	4	1.0	

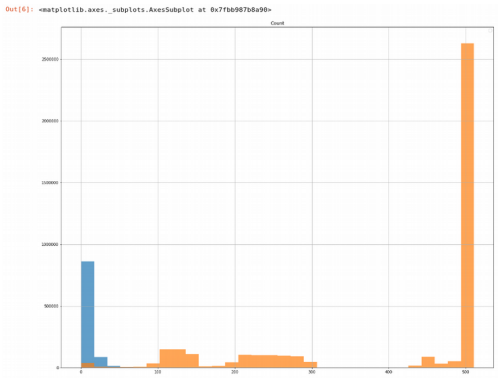
5 rows × 42 columns

```
In [26]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898431 entries, 0 to 4898430
Data columns (total 42 columns):
duration                        int64
protocol_type                   object
service                         object
flag                            object
src_bytes                       int64
dst_bytes                       int64
land                            int64
wrong_fragment                  int64
urgent                          int64
hot                             int64
num_failed_logins               int64
logged_in                       int64
num_compromised                 int64
root_shell                      int64
su_attempted                    int64
num_root                        int64
num_file_creations              int64
num_shells                      int64
num_access_files                int64
num_outbound_cmds               int64
is_host_login                   int64
is_guest_login                  int64
count                           int64
srv_count                       int64
serror_rate                     float64
srv_serror_rate                 float64
rerror_rate                     float64
srv_rerror_rate                 float64
same_srv_rate                   float64
diff_srv_rate                   float64

...
```

Posteriormente seleccionamos 3 atributos de los más valorados en el Ranking de selección y los visualizamos los datos en forma de ataque o normal en forma de histograma usando las funciones de matplotlib *(* En el anexo 2 se pueden ver los tres histogramas)*:

```
In [6]: pos_count = data[data['label'] == 'normal.']['count']
        neg_count = data[data['label'] != 'normal.']['count']

        pos_dst_host_same_src_port_rate = data[data['label'] == 'normal.']['dst_host_same_src_port_rate']
        neg_dst_host_same_src_port_rate = data[data['label'] != 'normal.']['dst_host_same_src_port_rate']

        pos_service = data[data['label'] == 'normal.']['service']
        neg_service = data[data['label'] != 'normal.']['service']

        fig = plt.figure(figsize=(20,200))

        #Count
        ax1 = fig.add_subplot(10,1,1)
        ax1.set_title("Count")
        ax1.legend()
        pos_count.hist(alpha = 0.7, bins = 30, label='positive')
        ax2 = fig.add_subplot(10,1,1)
        neg_count.hist(alpha = 0.7, bins = 30, label='negative')
```



Codificación de atributos

Para codificar los atributos categóricos a numéricos usamos la herramienta LabelEncoder de Scikit:

In [7]:
```
#TRANSFORMAR ATRIBUTOS categoricos EN NUMERO mediante from sklearn.preprocessing import LabelEncoder
number = LabelEncoder()

data_labels = data.label

data['protocol_type'] = number.fit_transform(data['protocol_type'].astype('str'))
data['service'] = number.fit_transform(data['service'].astype('str'))
data['flag'] = number.fit_transform(data['flag'].astype('str'))
data['label'] = number.fit_transform(data['label'].astype('str'))
```

Escalado de atributos

Posteriormente realizaremos el escalado de los datos, usando la función MinMax contenida en Scikit, indicando que queremos reducir el rango a valores entre 0 y 1:

In [9]:
```
#ESCALADO usando: from sklearn import preprocessing
minmax_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
data_minmax = minmax_scaler.fit_transform(data)
data_minmax = pd.DataFrame(data_minmax, columns=col_names)
```

Selección de atributos

Para la selección de atributos tomamos 10 los atributos que no nos interesan (calculados en el apartado 3 de la memoria) y eliminamos las columnas correspondientes de nuestro dataset.

In [10]:
```
y = data.label
X = data_minmax.drop('label', axis=1)
#X = data.drop('label', axis=1)
X = data_minmax.drop('is_host_login', axis=1)
X = data_minmax.drop('num_outbound_cmds', axis=1)
X = data_minmax.drop('urgent', axis=1)
X = data_minmax.drop('su_attempted', axis=1)
X = data_minmax.drop('num_shells', axis=1)
X = data_minmax.drop('land', axis=1)
X = data_minmax.drop('root_shell', axis=1)
X = data_minmax.drop('num_failed_logins', axis=1)
X = data_minmax.drop('num_file_creations', axis=1)
X = data_minmax.drop('num_root', axis=1)
#X = data_minmax.drop('is_guest_login', axis=1)
#X = data_minmax.drop('num_access_files', axis=1)
```

División del Dataset

Una vez hechas las transformaciones necesarias a nuestro dataset procederemos a dividirlo en dos subsets de train y test, y comprobar el tamaño de los sets resultantes:

```
In [12]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15)

In [13]: print("Training size: {}; Test size: {}".format(len(X_train),len(X_test)))
         X.shape

         Training size: 4163666; Test size: 734765

Out[13]: (4898431, 31)

In [15]: X_train.shape

Out[15]: (4163666, 31)

In [16]: X_test.shape

Out[16]: (734765, 31)
```

Entrenamiento del modelo:

Tras obtener los subsets de entrenamiento y test procedemos a entrenar nuestro algoritmo usando el arbol de decisión de Scikit (DecisionTreeClassifier). El único parámetro que configuraremos es “min_samples_split” que indica el mínimo número de muestras requeridas para dividir un nodo interno.

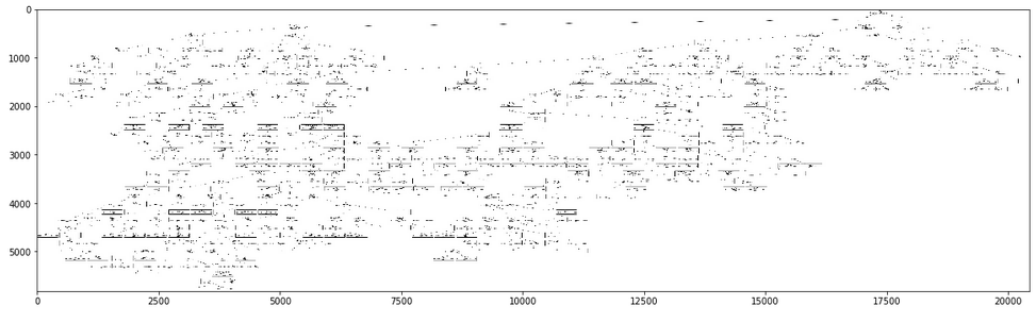
```
In [15]: from sklearn import tree #Arboles de decision
         from sklearn.tree import DecisionTreeClassifier, export_graphviz

         #El split marca la complejidad del arbol, si ponemos 2 quedaría lo mas complejo posible (over trained)
         c = DecisionTreeClassifier(min_samples_split=100)

In [16]: features = ["duration","protocol_type","service","flag","src_bytes",
             "dst_bytes","wrong_fragment","hot",
             "logged_in","num_compromised",
             "num_access_files",
             "is_guest_login", "count","srv_count","serror_rate",
             "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
             "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
             "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
             "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
             "dst_host_rerror_rate","dst_host_srv_rerror_rate"]

In [17]: print time.strftime("%H:%M:%S")
         dt = c.fit(X_train, y_train)
         print time.strftime("%H:%M:%S")

         14:11:01
         14:11:48
```

Una vez entrenado nuestro algoritmo procedemos a mostrar el arbol de decisión mediante la función propuesta en el siguiente código (Se anexará al proyecto una imagen exportada de alta resolución con la solución final).

```
In [18]: def show_tree(tree, features, path):
             f = io.StringIO()
             export_graphviz(tree, out_file=f, feature_names=features)
             pydotplus.graph_from_dot_data(f.getvalue()).write_png(path)
             img = imageio.imread(path)
             plt.rcParams["figure.figsize"] = (20,20)
             plt.imshow(img)
```

```
In [22]: show_tree(dt, features, 'dec_tree_01.png')
```

```
/usr/local/lib/python2.7/dist-packages/PIL/Image.py:2514: DecompressionBombWarning: Image size (118858411 pixels) exceeds
limit of 89478485 pixels, could be decompression bomb DOS attack.
  DecompressionBombWarning)
```



Cálculo de la precisión

Por último podemos realizar la predicción con el subset de test, y calcular la precisión de nuestro algoritmo:

```
In [20]: #Predicciones con la parte de dataset de test
         y_pred = c.predict(X_test)
```

```
In [21]: from sklearn.metrics import accuracy_score
         score = accuracy_score(y_test, y_pred) * 100
         print "Accuracy using Decision Tree: ", score
```

```
Accuracy using Decision Tree:  99.9810823868856
```

Resultados finales:

Tiempo de entrenamiento del modelo: 49 Segundos

Precisión: 99.984%

5.2. Propuesta de mejoras

Partiendo de la solución propuesta se muestran resultados de otras configuraciones para demostrar el impacto de cada paso en el resultado final, y buscar posibles mejoras. Cada modificación se realiza de manera independiente, y luego se aplican todas las modificaciones que han resultado relevantes:

1. Modificación: Clasificación binaria (ataque o normal)

Agregaremos las siguientes lineas a nuestro código para modificar el dataset

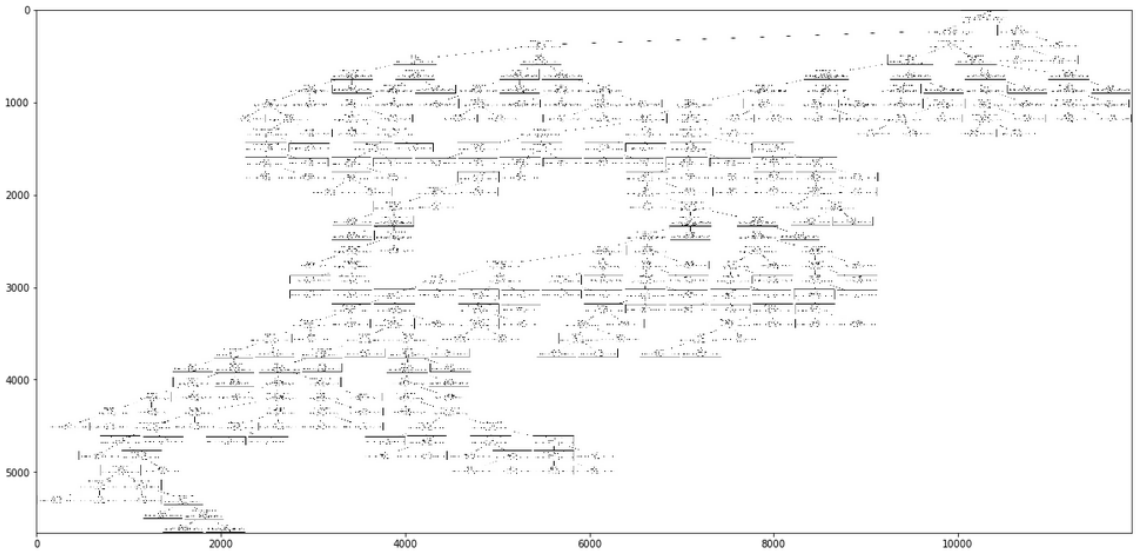
```
In [7]: #Llamar attack a los que no sean normales
        data.loc[data['label']!='normal.','label'] = 'attack.'
```

Tiempo entrenando modelo: 49 Segundos

Precisión: 99,984%

Conclusión: El tiempo de entrenamiento es prácticamente el mismo, aumenta ligeramente la precisión, pero no distinguimos el tipo de ataque.

2. Modificación: Reducción del árbol (aumentamos min_samples_split de 100 a 1000)

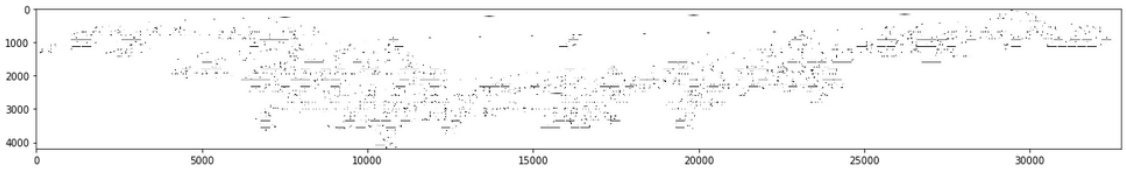


Tiempo entrenando modelo: 47 Segundos

Precisión: 99,934%

Conclusión: Se pierde precisión, no cambia el tiempo de entrenamiento.

3. Modificación: Aumento del árbol (reducimos min_samples_split de 100 a 10)



Tiempo entrenando modelo: 47 Segundos

Precisión: 99,988%

Conclusión: Se gana ligeramente precisión, no cambia el tiempo de entrenamiento.

*Se intenta reducir el valor de min_samples_split al mínimo 2 y se obtiene una precisión menor (sobre entrenamiento).

4. Modificación: No aplicar normalización de atributos

```
In [10]: #ESCALADO usando: from sklearn import preprocessing
         #minmax_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
         #data_minmax = minmax_scaler.fit_transform(data)
         #data_minmax = pd.DataFrame(data_minmax, columns=col_names)
```

Tiempo entrenando modelo: 44 Segundos

Precisión: 99,986%

Conclusión: Se gana ligeramente precisión, se obtuvo menor tiempo de entrenamiento.

5. Modificación: Eliminar dos atributos más (teniendo en cuenta resultados en el ranking)

In [*]:
```
y = data.label
X = data_minmax.drop('label', axis=1)
#X = data.drop('label', axis=1)
X = X.drop('is_host_login', axis=1)
X = X.drop('num_outbound_cmds', axis=1)
X = X.drop('urgent', axis=1)
X = X.drop('su_attempted', axis=1)
X = X.drop('num_shells', axis=1)
X = X.drop('land', axis=1)
X = X.drop('root_shell', axis=1)
X = X.drop('num_failed_logins', axis=1)
X = X.drop('num_file_creations', axis=1)
X = X.drop('num_root', axis=1)
X = X.drop('is_guest_login', axis=1)
X = X.drop('num_access_files', axis=1)
```

Tiempo entrenando modelo: 48 Segundos

Precisión: 99,978%

Conclusión: Se pierde precisión, mismo tiempo de entrenamiento.

6. Modificación: Eliminar sólo 7 atributos (teniendo en cuenta resultados en el ranking)

Tiempo entrenando modelo: 48 Segundos

Precisión: 99,979%

Conclusión: Se pierde precisión, mismo tiempo de entrenamiento.

7. Modificación: No eliminar atributos

In [22]:
```
y = data.label
X = data_minmax.drop('label', axis=1)
#X = data.drop('label', axis=1)
#X = X.drop('is_host_login', axis=1)
#X = X.drop('num_outbound_cmds', axis=1)
#X = X.drop('urgent', axis=1)
#X = X.drop('su_attempted', axis=1)
#X = X.drop('num_shells', axis=1)
#X = X.drop('land', axis=1)
#X = X.drop('root_shell', axis=1)
#X = X.drop('num_failed_logins', axis=1)
#X = X.drop('num_file_creations', axis=1)
#X = X.drop('num_root', axis=1)
#X = X.drop('is_guest_login', axis=1)
#X = X.drop('num_access_files', axis=1)
```

Tiempo entrenando modelo: 55 Segundos

Precisión: 99,979%

Conclusión: Se pierde precisión, aumenta el tiempo de entrenamiento.

5.3. Solución definitiva

Una vez analizados varios escenarios, se decide realizar las siguientes modificaciones (Se puede consultar el código completo en el archivo adjunto al proyecto Anexo2.pdf, o TFM.ipynb en formato jupyter notebook):

- Se elimina la normalización
- Se aumenta el tamaño del árbol de decisión

Resultados finales:

Se ha conseguido reducir ligeramente el tiempo de entrenamiento, y aumentar la precisión hasta un valor muy bueno, mayor al 99,9916%.

Tiempo entrenando modelo: **45 Segundos**

Precisión: **99,9916%**

```
In [20]: from sklearn.metrics import accuracy_score
         score = accuracy_score(y_test, y_pred) * 100
         print "Accuracy using Decision Tree: ", score

         Accuracy using Decision Tree:  99.99169802589944
```

En caso de hacer una **clasificación binaria** el resultado es todavía mejor, **99,9955%.**

```
In [19]: from sklearn.metrics import accuracy_score
         score = accuracy_score(y_test, y_pred) * 100
         print "Accuracy using Decision Tree: ", score

         Accuracy using Decision Tree:  99.99550876810953
```

6. CONCLUSIONES

6.1. Consecución de objetivos

En este apartado realizamos un análisis de objetivos, y su grado de consecución, así como posibles variaciones en la planificación temporal prevista inicialmente.

Objetivos del trabajo:

- Exponer la utilidad del aprendizaje automático, y particularmente su aplicación a la seguridad.

  Conseguido: Se ha hecho un análisis detallado de la historia del Machine Learning, y su aplicación a la seguridad, comparándolo con los sistemas tradicionales.

- Estudiar herramientas de aprendizaje automático.

  Conseguido: Se ha aprendido a usar diversas opciones de Weka, y Scikit-Learn.

- Comparar la precisión de diferentes algoritmos para un caso particular.

  Conseguido: Se ha realizado el análisis correspondiente con Weka y Scikit-learn.

- Realizar un tratamiento de los datos con el fin de optimizar el proceso de aprendizaje.

  Conseguido: Se han realizado distintas modificaciones al dataset basándose en una versión inicial del código, y se han realizado diversos ajustes a las mismas, para obtener los mejores resultados en la solución final.

- Optimizar los parámetros de entrenamiento y del modelo para conseguir la máxima precisión.

  Conseguido: Se han probado diferentes valores del parámetro min_samples_split, eligiendo la que mejor resultado daba, obteniendo un 99,9916% de precisión.

- Obtener un script que permita realizar todo el proceso automáticamente con los mejores resultados posibles.

  Conseguido: Se ha realizado un script en Python que utiliza un árbol de decisión para detectar los ataques, con unos resultados finales muy buenos.

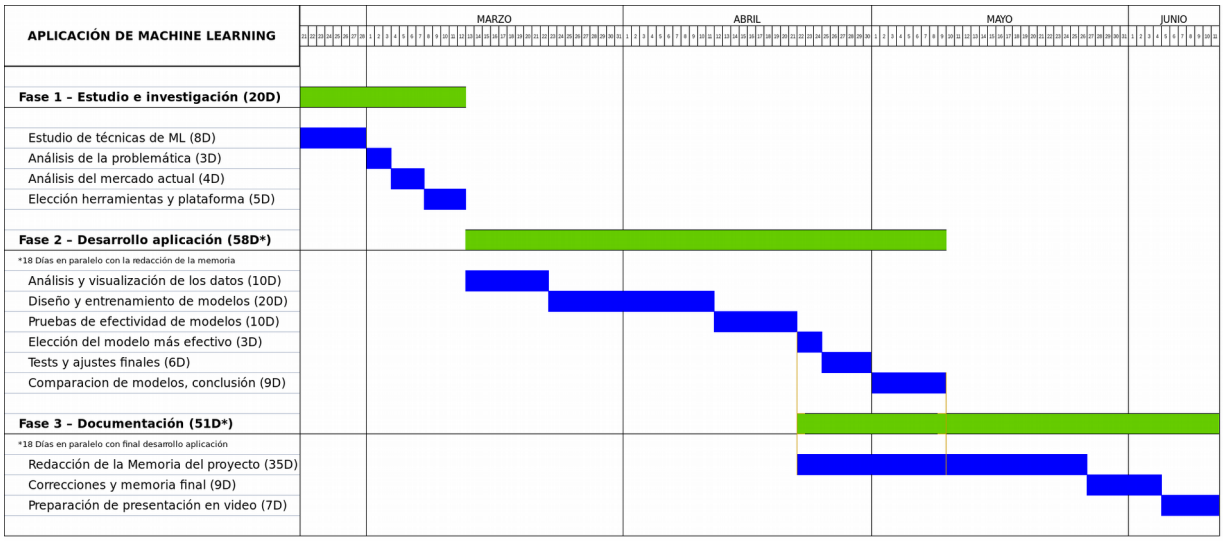
Planificación:

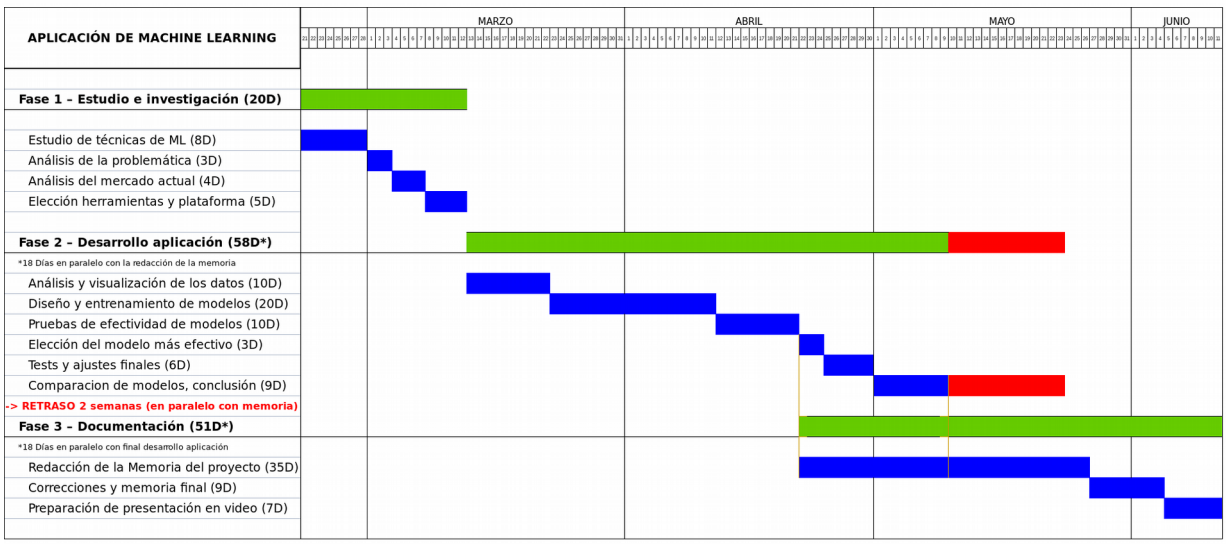
Debido a la falta de conocimiento y experiencia en la materia, se han perdido varios días ya que el script daba resultados de precisión muy altos desde la primera versión del código, lo cual condujo a la sospecha de que habría un error en alguna parte.

Tras realizar pruebas con Weka (herramienta que no estaba prevista en un principio) y leer diversos documentos acerca del dataset KDD99, se dedujo que estos resultados eran normales para el dataset propuesto.

Este error ocasionó una pérdida considerable del tiempo disponible para la redacción de la memoria al solaparse durante 2 semanas más la realización del script, ya que se tuvo que investigar y probar otras herramientas con el fin de contrastar los resultados obtenidos.

A continuación se muestran los dos cronogramas (previsto y real):





6.2. Autoevaluación

Me ha resultado difícil adquirir los conocimientos necesarios para realizar el proyecto, ya que empezaba de cero en esta disciplina, y no conocía ninguna herramienta ni los procesos básicos. Además el error antes comentado de presuponer que los algoritmos no podrían llegar a ser tan precisos, hizo que perdiera mucho tiempo, que podría haber aprovechado para realizar un trabajo más técnico y profundizar más en ciertos apartados, y quizás aventurarme a comparar con otros algoritmos más complejos.

Si bien el proceso de formación me ha resultado complicado, una vez comprendidos los conceptos básicos considero que supe aprovechar lo aprendido, y pude conseguir un resultado con la aplicación final que es mejor de lo que esperaba.

He aprendido a usar dos herramientas muy interesantes y además he adquirido mucho conocimiento en la materia, que me permite tener una buena base para seguir formándome.

7. GLOSARIO

clasificación binaria (binary classification) Tipo de tarea de predicción que da como resultado una de dos clases mutuamente exclusivas.

datos categóricos (categorical data) Atributos que tienen un conjunto discreto de valores posibles.

clase (class) Valor de un conjunto de valores de segmentación enumerados para una etiqueta.

modelo de clasificación (classification model) Tipo de modelo de aprendizaje automático para distinguir entre dos o más clases discretas.

Instancia, ejemplo. Fila de un conjunto de datos. Un ejemplo contiene uno o más atributos y, posiblemente, una etiqueta. Consulta también ejemplo etiquetado y ejemplo sin etiqueta.

regresión logística (logistic regression) Modelo que genera una probabilidad para cada valor de etiqueta discreta posible en problemas de clasificación al aplicar un atributo sigmoide a una predicción lineal.

modelo (model) Representación de lo que un sistema aprendió de los datos de entrenamiento.

entrenamiento de modelos (model training) Proceso de determinar el mejor modelo.

clasificación de múltiples clases (multi-class classification) Problemas de clasificación que distinguen entre más de dos clases.

normalización (normalization) Proceso de convertir un rango real de valores en un rango estándar de valores, generalmente -1 a +1 o 0 a 1.

datos numéricos (numerical data) Atributos representados como números enteros o de valores reales.

sobreajuste, sobreentrenamiento (overfitting) Creación de un modelo que coincide con los datos de entrenamiento tan estrechamente que el modelo no puede realizar predicciones correctas con datos nuevos.

rendimiento (performance) Rapidez de ejecución de un software

precisión (precision) Métrica para los modelos de clasificación. La precisión identifica la frecuencia con la que un modelo predijo correctamente la clase positiva.

predicción (prediction) Resultado de un modelo cuando se le proporciona un ejemplo de entrada.

rango (rank) La posición ordinal de una clase en un problema que categoriza clases de la más alta a la más baja.

modelo de regresión (regression model) Tipo de modelo que da como resultado valores continuos (generalmente de punto flotante).

ajuste (scaling) Práctica que se usa comúnmente en la ingeniería de atributos para acotar el rango de valores de un atributo a fin de que coincida con el rango de los otros atributos en el conjunto de datos.

aprendizaje semisupervisado (semi-supervised learning) Entrenamiento de un modelo sobre datos en el que algunos de los ejemplos de entrenamiento tienen etiquetas, pero otros no.

aprendizaje automático supervisado (supervised machine learning) Entrenamiento de un modelo a partir de datos de entrada y sus etiquetas correspondientes.

Target. Sinónimo de etiqueta.

conjunto de prueba (test set) Subconjunto del conjunto de datos que se usa para probar un modelo.

entrenamiento (training) Proceso de determinar los parámetros ideales que conforman un modelo.

conjunto de entrenamiento (training set) Subconjunto del conjunto de datos que se usa para entrenar un modelo.

aprendizaje por transferencia (transfer learning) Transferencia de información de una tarea de aprendizaje automático a otra.

ejemplo sin etiqueta (unlabeled example) Ejemplo que contiene atributos, pero no etiqueta.

aprendizaje automático no supervisado (unsupervised machine learning) Entrenamiento de un modelo para encontrar patrones en un conjunto de datos, generalmente sin etiqueta.

peso (weight) Coeficiente para un atributo en un modelo lineal o una conexión en una red profunda.

8. BIBLIOGRAFÍA Y RECURSOS

Intrusion Detection in KDD99 Dataset: A Review

Megha Jain Gowadiya, Anurag Jain

https://www.ijser.org/researchpaper/Intrusion-Detection-in-KDD99-Dataset-A-Review.pdf

Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD99 Intrusion Detection Datasets

H. Günes Kayacık, A. Nur Zincir-Heywood, Malcolm I. Heywood

https://pdfs.semanticscholar.org/1d6e/a73b6e08ed9913d3aad924f7d7ced4477589.pdf

Scikit-learn y librerías de Python

https://en.wikipedia.org/wiki/Scikit-learn

http://scikit-learn.org/stable/

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

Weka

https://es.wikipedia.org/wiki/Weka_(aprendizaje_autom%C3%A1tico)

https://machinelearningmastery.com/perform-feature-selection-machine-learning-data-weka/

Machine Learning

https://www.inbest.cloud/comunidad/8-aplicaciones-de-machine-learning

https://planetachatbot.com/inteligencia-artificial-ai-estado-del-arte-67b325c7bbfc

https://towardsdatascience.com/ai-and-machine-learning-in-cyber-security-d6fbee480af0

http://cso.computerworld.es/cibercrimen/cinco-casos-de-uso-de-machine-learning-en-seguridad

Ingeniería de atributos, codificación, normalización

https://developers.google.com/machine-learning/crash-course/representation/feature-engineering?hl=es-419

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

https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/

http://pbpython.com/categorical-encoding.html

Selección de atributos

https://docs.microsoft.com/es-es/azure/machine-learning/studio/algorithm-choice

Definiciones

https://developers.google.com/machine-learning/crash-course/glossary?hl=es-419#p

9. ANEXOS

Se han realizado 2 anexos y 3 archivos que se adjuntarán al proyecto. El listado es el siguiente:

- ANEXO 1: Definición y gráfico de todos los atributos (pdf)
- ANEXO 2: Código y ejecución de la Aplicación (pdf)
- Código de la aplicación en formato ipynb (Jupyter-notebook)
- Imagen de alta resolución con el árbol de decisión
- Datasets 10% en formato ARFF (Weka) y CSV.