Aprendizaje Automático para la detección de ataques informáticos

Antonio Herrera Zurita

Junio 2016

Resumen–
En la actualidad prácticamente todo el mundo conoce Internet y dispone de él en casa, o en su empresa, organización, etc. Sin embargo, en la red no solo hay usuarios corrientes con intenciones buenas, también los hay que buscan lucrarse, molestar, o dañar a otros. Eso provoca que Internet no sea seguro y se produzcan muchos ciberataques que pueden afectar tanto a usuarios corrientes como a organizaciones, empresas o instituciones.
Así pues, y bajo esa premisa, los ciberataques, lo que se presenta en el siguiente artículo es la de explotar una rama relativamente reciente y en crecimiento de la Inteligencia Artificial como es el Aprendizaje Automático de las maquinas. Esta nueva rama puede usarse en muchas facetas de la vida cotidiana.
Por tanto, lo que se va a exponer es una breve descripción de que es el Aprendizaje Automático, que aplicaciones se le pueden a dar y, en particular nos centraremos en la de la predicción de un tipo de ataque maliciosos como son las producidas por los accesos a URLs maliciosas.

Palabras clave– Aprendizaje automático, características, set de datos, host-based, características lexicas, precisión, Bayesiano Ingenuo, K-vecinos más cercanos, URL, lista negra.

Abstract– Nowadays almost everybody knows or has Internet at home, in the enterprise, organization, etc. At a same time, not only current users use Internet. There are some fraudulent users who attempt to others users or commit crimes. This provoke that every day lots of cyber-attacks occurs to any kind of user.
Taking this at a premise, the intent of this paper is to exploit a relative new branch in the Artificial Intelligence called Machine Learning. Machine Learning is a set of techniques an algorithms that are in continuous expansion in multiple domains such as preventing cyber-attacks.
In this paper it will be exposed a short introduction to Machine Learning, its uses and how can this techniques and algorithms can help to detect and mitigate a type of attack which is based on the access to a malicious URL.

Keywords– Machine Learning, features, dataset, host-based, lexical features, accuracy, Naive Bayes, K-Nearest Neighbours, URL, Blacklist.

1 Introducción

En la actualidad prácticamente todo el mundo conoce Internet y dispone de él en casa. Actualmente hay más de 7 billones de usuarios de Internet. [1] La mayoría de usuarios lo utilizan para tareas domésticas, institucionales o comerciales. Aun así existe una minoría que utiliza internet para lucrarse a costa de otros o comprometerles el sistema, robarles información, etc. Es gente que se dedica a comprometer dispositivos ajenos y/o a delinquir, ya sea por diversión, por dinero, etc. Así pues hoy en día existen todo tipo de ataques a usuarios, compañías, organizaciones, etc.

Actualmente, y según el Centro Nacional de Protección de Infraestructuras Críticas, España se sitúa como el tercer país que más ciberataques sufre. A lo largo del día se pro-

• E-mail de contacte: antonio.herrera@uab.cat
• Menció realitzada: Enginyeria de Tecnologies de la Informació
• Treball tutoritzat per: Jordi Duran Cals (Departament d'Enginyeria de la Informació i de les Comunicacions Ciències de la computació i la Intel·ligència Artificial)
• Curs 2015/16

ducen entre 180.000 y 200.000 ciberataques. Algunos de los tipos de ataques más frecuentes son los que se listan a continuación: [2]

1. Adware: los molestos anuncios.
2. Malware y ransomware : virus cuyo objetivo es alterar el funcionamiento de un dispositivo
3. Spyware: un espía muy peligroso el cual recopila información y se lo manda a terceras personas externas al dispositivo.
4. Gusanos y troyanos: los cuales pueden replicarse, dar acceso remoto a un atacante.
5. Phising: los cuales intentan convencerte de hacer clic en links o para ingresar datos confidenciales.
6. Backdoors: La puerta trasera es una secuencia especial mediante la cual se pueden evitar los sistemas de seguridad del ordenador.

De todos los tipos de ataques que se listan y que existen para este proyecto nos centraremos en el de comprometer un dispositivo de cualquier forma al acceder a una URL maliciosa ya que la gran mayoría de esos ataques se pueden producir simplemente accediendo a una URL fraudulenta.

Por tanto debemos entender que este tipo de ataque puede ser muy serio y no debemos subestimarlo. Este tipo de ataque puede comprometer muy seriamente nuestro sistema por el simple hecho de que ese acceso puede desencadenar que alguno o algunos de los que hemos mencionado se perpetúen.

Por otro lado y según las nuevas tendencias y descubrimientos, lo que se pretende de los sistemas es que cada vez sean más inteligentes y ayuden a las personas en su día a día ya sea con un asistente virtual que te ayude con problemas con tu banco, robots que asistan a personas con movilidad reducida, o sistemas que sean capaces de aprender automáticamente.

Detrás de muchos de éstos sistemas inteligentes se esconde una nueva rama de la Inteligencia Artificial llamada **Aprendizaje Automático** o, en inglés **Machine Learning**. Más adelante se definirá que es el aprendizaje automático pero de momento, nos basta con saber que hay muchas técnicas y algoritmos de ésta rama.

Por tanto, en este artículo se va a hacer una exploración de las de las técnicas del Aprendizaje automático y se utilizará este tipo de ataques mediante URL como base de los experimentos y las pruebas.

Los objetivos de este proyecto son tomando como premisa este ciberataque, utilizar y explorar técnicas del Aprendizaje Automático para la creación de un modelo capaz de predecir éste tipo de ataques.

Concretamente lo que se expondrá será la metodología de desarrollo que se siguió para la generación de un modelo de aprendizaje entrenado mediante técnicas de Aprendizaje Automático con la intención de prevenir este tipo de ataques.

Para ello se nos presentaron una serie de escenarios los cuales explicaremos a continuación. Se explicará en qué consistía cada uno de ellos, cuáles eran sus objetivos, sus planificaciones y se expondrá cual fue finalmente la opción elegida junto a su justificación y cómo se llevó a cabo.

1.1. Escenario 1. Herramienta de detección de URL.

En un principio, el objetivo de este proyecto consistía en crear una herramienta que predijera este tipo de ataques. Se desarrollaría una herramienta que ante una URL cualquiera, fuera capaz de detectar si era maliciosa o no.

Para ello, en un principio se creía que se dispondría de unos **datasets** ya hechos los cuales estarían listos para utilizar.

Un dataset no es más que un conjunto de datos con una cierta estructura. En nuestro caso, estos datasets contenían tanto URLs maliciosas como benignas y se pretendían utilizar como fuente de prueba y experiencia del modelo. Esos conjuntos de datos consistían en unas dieciséis mil URLs tanto buenas como malas. Por cada registro del dataset, es decir, por cada URL podían haber asociadas hasta más de tres millones de features.

Las features o características, son propiedades asociadas a cada URL las cuales ayudarán al modelo a identificar ante una determinada URL si ésta es perniciosa o no.

En la siguiente subsección explicaremos los objetivos que tenía este escenario:

1.1.1. Objetivos del escenario 1.

Los objetivos de este primer escenario eran:

- Crear una herramienta que analice si una URL es perjudicial o no.
- Explorar técnicas para la extracción de features de una URL.
- Explorar diferentes técnicas de Aprendizaje Automático para el entrenamiento de un modelo capaz de clasificar una URL.
- Obtener y exponer el conocimiento aprendido.
- Mostrar la herramienta en funcionamiento y comprobar que realmente era capaz de identificar si una URL era maliciosa o no.

El inconveniente que surgió fue que esas features, estaban recolectadas de manera que no era posible identificar a qué correspondía cada una de éstas. Esas características estaban en formato numérico y por tanto, no había manera de descifrar a qué pertenecía cada valor. Además habían sido recolectadas mediante un recolector de características al cual no se tenía acceso.

Por lo tanto, aunque era posible entrenar un modelo con esos datos no se era capaz de, ante una URL cualquiera extraer esas mismas características que identificaban la URL y establecer una correlación que ayudara a la herramienta a identificar si la URL era benigna o perjudicial.

Como no tenía sentido extraer unas características al azar y relacionarlas con las del dataset, para el cumplimiento del objetivo principal del escenario 1, había que descartar la utilización de esos datasets y generarse uno propio el cual contuviera tanto un conjunto de URLs benignas y maliciosas como features extraídas manualmente.

El problema de este escenario y el motivo principal por el que se descartó fue que el foco principal que tenia este proyecto era en la extracción de características de las

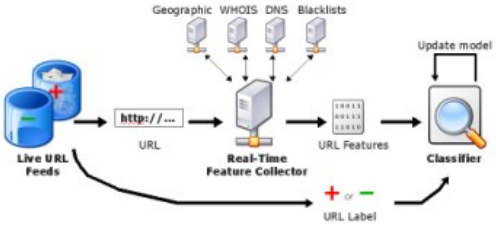

Fig. 1: Escenario 1

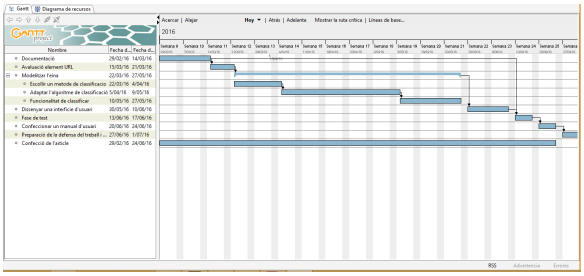

Fig. 2: Planificación Inicial Escenario 1

URLs más que en el estudio de técnicas del Aprendizaje Automático o incluso de creación de un modelo.

Como ya hemos mencionado anteriormente, las características(features) relacionadas a cada URL son las que nos ayudan a determinar como de diferente tiende a ser una URL maliciosa de una que no lo es. Así pues, de estas características es de lo que se tiene que alimentar nuestro modelo.

Obtener y extraer features de las URLs no es sencillo ya que no solo se deben obtener las **Host-Based features** como pueden ser la geolocalización del host, el host, la IP perteneciente a la URL o incluso identificarla dentro de una Blacklist. Una blacklist es una lista negra de URLs que ya han sido identificadas y etiquetadas como URLs con contenido malicioso de cualquier tipo [3].

En cualquier caso, también se debían extraer un conjunto de características llamadas **Lexical features**. Las características léxicas o también denominados conjuntos de palabras son agrupaciones de palabras contenidas en las URLs usando los delimitadores propios de las mismas para establecer el conjunto de palabras que se quiere conseguir [4].

Veamos un ejemplo, en la URL www.geocities.com/usr/index.html podemos identificar las dos partes claramente. Por un lado www.geocities.com haría referencia a la parte de host mientras que /usr/index.html haría referencia a la parte del camino. Las características léxicas de las URLs se extraen de esta segunda parte.

Imaginemos que en la parte del camino hubiera un directorio que fuera /virus/index.html. En ese caso, podríamos obtener una agrupación de palabras que fuera ”virus”para identificar que cualquier URL que contenga eso en su URL es potencialmente una URL maliciosa.

Para este escenario se realizó una serie de planificaciones que en la siguiente subsección se comentaran.

1.1.2. Planificación del escenario 1

Para este escenario se realizó una planificación la cual se muestra en la figura 3.

Como se puede ver, en esta primera planificación en ningún momento aparece la tarea de generación de datasets. Eso no está reflejado por que como hemos comentado se contaba con que se podría disponer de unos datasets de los cuales finalmente no se podía. Lamentablemente eso no se contempló hasta más adelante. Por tanto, cuando se descubrió que esos datasets no iban a servir para el objetivo principal del proyecto, la planificación cambió a la de la mostrada en la siguiente figura.

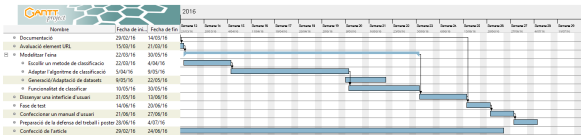

Fig. 3: Planificación Actualizada del Escenario 1

En esta planificación sí que está contemplada la tarea de generar los datasets propios. La idea era conseguir unos datasets generados manualmente que sirvieran para entrenar al modelo y así ante una nueva URL extraer las mismas features extraídas para el conjunto de datos y poder hacer una correlación con el objetivo de predecir si esa nueva URL era potencialmente maliciosa o no.

Debido a que no era ese el enfoque que se le quería dar al proyecto, éste escenario se descartó y se le dio un nuevo enfoque el cual partiera desde la misma base y objetivos. Ahí surgió el escenario 2 que se explicará a continuación y el que finalmente se implementó.

1.2. Escenario 2. Generación y comparación de modelos de Aprendizaje automático basados en modelos probabilísticos.

El escenario 2 mantenía el mismo objetivo de explorar técnicas del Aprendizaje Automático para la prevención de ataques producidos por el acceso a URLs fraudulentas pero dándole una perspectiva diferente. En vez de realizar una herramienta capaz de predecir ataques, lo que se pretendía era generar un modelo entrenado mediante datos reales con alguna de las técnicas estudiadas y que se explicarán durante la sección del estado del arte. De esta manera se podría:

- Ver la importancia de los datos con los que se alimentan este tipo de modelos.
- Ver como a medida que el modelo se entrena con más datos la precisión no necesariamente aumenta.
- Establecer una metodología para evaluar los conjuntos de características para entrenar un modelo estadístico.

- Comparar los resultados de dos modelos probabilísticos y ver cual de los dos ofrece unos mejores resultados para éste caso.
- Alimentar el modelo con datos que realmente ayuden al modelo a mejorar en el entrenamiento.
- Hacer un entrenamiento exhaustivo del modelo y conseguir la mejor precisión posible.
- Notificar que cuando se manejan tantos datos la potencia de cómputo es importante.
- Generación de otro modelo probabilístico mediante otro algoritmo para realizar pruebas y una comparación entre ambos.
- Obtener y exponer el conocimiento aprendido.

En este caso, usamos los datasets proporcionados por Secrepo [5]. Y tomando como fuente de experiencia esos datos se podría generar el modelo que se buscaba el cual estaría entrenado con datos reales y el cual podría ser testeado con datos con los que no había sido entrenado y sacar así unas métricas para ver cuán efectivo era el modelo.

Para el entrenamiento de este modelo tuvimos que hacer una adaptación previa de los datasets ya que estos estaban en formato ”svm”.

Esto es así porque los datasets estaban preparados para ser introducidos en un programa llamado ”SVM-light”.

SVM-light es la implementación de Support Vector Machine en C [6]. Esta herramienta a la hora de realizar el entrenamiento omite las columnas del documento de entrada que son completamente 0. Así pues, los datasets estaban en un formato en el que solo estaban expresadas las características cuyo valor fuera diferente de 0.

Este tipo de archivos con este formato se utiliza básicamente ya que al trabajar con tal volumen de datos estos archivos pueden llegar a ocupar mucho espacio y mucha memoria al utilizarlos. De esta manera se simplifica el peso de este tipo de archivos y los hace más computables.

En todo caso, el algoritmo que nosotros usamos necesitaba como parámetro de entrada un archivo en “csv”. CSV es un formato de Excel el cual delimita las columnas con comas. Éste formato sí que necesitaba expresar las columnas a en la posición que correspondía. Por tanto, era necesario hacer una adaptación.

Para que sea más claro de entender, a continuación en la siguiente figura mostraremos el archivo que teníamos antes de convertir y lo analizaremos para ver cómo se adaptó.

Fig. 4: Dataset inicial

Como se puede ver en la figura 4, la primera columna corresponde al tipo de clase de la URL. En éste caso se tenían dos clases diferentes. La clase -1 significaba que la URL era maliciosa, y 1 significaba que era una URL benigna.

Para el uso de los datasets con la herramienta de SVM-light ya funcionaba correctamente. En cambio, el algoritmo que se utilizó y del que hablaremos posteriormente, solo aceptaba enteros positivos y, además, esperaba la clase al final. Por tanto, todo registro cuya clase tuviera un -1, pasó a ser 0 y dejó de estar en la primera posición para pasar a la última columna del dataset.

El resto, como se puede apreciar en la figura, contienen la columna y su correspondiente valor expresado de la siguiente forma “4:0.78”. Por tanto, para la adaptación, se usó el primer valor el cual indicaba en que columna del csv se debía que escribir seguido del valor que le correspondía.

De esta manera lo que se obtuvo era un csv adaptado al algoritmo. En la figura 5 se puede ver el dataset adaptado.

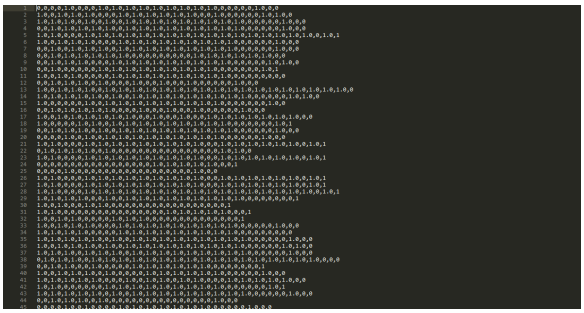

Fig. 5: Dataset Adaptado

Una vez se tenían los datos la siguiente tarea era la de construir el modelo. Para ello, se necesitaba un algoritmo con el que entrenar dicho modelo usando los datos como la fuente de experiencia. El algoritmo, en el cuál entraremos más en detalle posteriormente fue Naive Bayes, al cual solo se le hizo un cambio para la corrección de la muestra que se detallará el por qué más adelante.

Naturalmente, este nuevo escenario debía tener una nueva planificación. La nueva planificación es la que se muestra a continuación. En la siguiente subsección se comentará dicha planificación.

1.2.1. Planificación del escenario 2

La siguiente figura muestra la planificación de este nuevo escenario y difiere un poco del escenario 1 aunque a grandes rasgos sigue siendo muy similar.

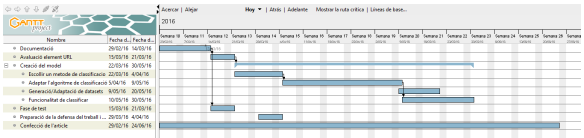

Fig. 6: Planificación Escenario 2

Como se puede apreciar en esta nueva planificación la tarea principal ha pasado de ser la de “Creación de una herramienta” a la de creación de un modelo. Además se añadió una tarea de **Adaptación de los datasets**, se eliminó la de la Interfaz de usuario y la de la confección de un manual de usuario pues no tenían sentido.

Explicados los dos tipos de escenarios, como ya hemos dicho anteriormente en este artículo nos vamos a centrar en esta nueva faceta de aprendizaje automática que ayude

a mitigar este tipo de ataques maliciosos siguiendo el escenario 2 y realizar pruebas exhaustivas para ver determinar una gráfica de resultados en función del incremento de características así como una comparación con otro algoritmo basado también en un modelo probabilístico.

Para ello vamos a estructurar este documento en una serie de secciones. Primero de todo se comentará muy rápidamente en qué consiste el Aprendizaje Automático y que usos se le puede dar aunque nos centraremos en la de prevención de ciberataques. Se comentarán los tipos de algoritmos que se pensaron en utilizar y de los cuales se documentó. De esos algoritmos se seleccionaron dos los cuales veremos más adelante cuales fueron y por qué. Seguidamente se explicará la metodología de desarrollo seguida, como se estructuró el proyecto, que lenguajes de programación se utilizaron y qué artefactos se generaron. Posteriormente, se explicarán los resultados obtenidos de la creación, entrenamiento y testeo del modelo generado. Se realizará una comparación sobre los dos algoritmos seleccionados para ver las diferentes precisiones y tiempos obtenidos. Y para finalizar, en la sección de las conclusiones se extraerán las conclusiones que se han podido obtener una vez finalizado el proyecto.

2 Estado del arte

Como hemos comentado para la realización de este trabajo se utilizó Aprendizaje Automático. A continuación haremos una breve explicación de lo que es el Aprendizaje Automático y algunos de sus usos. Por último también analizaremos un poco los algoritmos que se estudiaron para la realización del proyecto.

2.1. Qué es el Aprendizaje Automático?

Para las ciencias de la computación, el Aprendizaje Automático es una rama de la Inteligencia Artificial [7] que dote a las máquinas de técnicas de aprendizaje. Concretamente se trata de crear programas capaces de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos. Hay varios modelos de Aprendizaje Automático:

- Modelo geométrico: construidos en el espacio de instancias y que pueden tener una, dos o múltiples dimensiones.
- Modelo probabilístico: que intentan determinar la distribución de probabilidades descriptora de la función que enlaza a los valores de las características con valores determinados.
- Modelo lógico: expresan las probabilidades en reglas organizadas en forma de árboles de decisión. [8]

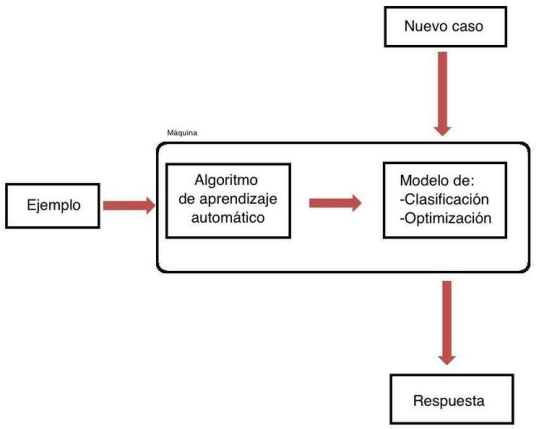

Fig. 7: Esquema Aprendizaje Automático

2.1.1. Usos del Aprendizaje Automático

Los usos del Aprendizaje Automático son muy numerosos debido a la gran versatilidad que tiene. Por tanto no nos extenderemos mucho en esto y listaremos simplemente para qué se podría utilizar el Aprendizaje Automático además de para la predicción de ataques. [9]

- Detección de correos molestos (SPAM)
- Detección de patrones en imágenes
- Reconocimiento de caracteres
- Recomendación de productos

2.2. Algoritmos de entrenamiento

Inicialmente para la realización de éste trabajo se barajaron una serie de de algoritmos que podrían servir. En consecuencia, se hizo una documentación previa sobre tres de ellos. Estos eran **Naive Bayes**, **Support Vector Macine** y **Logaritmic Regression**. Posteriormente y más avanzado en el proyecto se estudió otro algoritmo llamado **K-Nearest Neighbors**. Hagamos una breve introducción a cada uno de ellos:

- Naive Bayes
- Support Vector Machine (SVM)
- Regresión Logística (Logistic Regression)
- K-nearest Neighbors(KNN)

2.2.1. Naive Bayes

Consiste en una técnica de clasificación y predicción supervisada que construye modelos que predicen la probabilidad de posibles resultados.

Al ser una técnica de aprendizaje supervisado necesita de ejemplos que constituyan la base de conocimiento y experiencia. Está basado en el teorema de Bayes. No entraremos en detalle en el teorema de Bayes pero para que nos hagamos una idea, este teorema también puede ser llamado el teorema de la probabilidad condicionada ya que se calculan probabilidades de que ocurra un suceso que afecta a otro. [10]

2.2.2. Support Vector Machine (SVM)

Las máquinas de soporte vectorial son un conjunto de algoritmos de aprendizaje supervisado. Estos métodos están propiamente relacionados con problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento (de muestras) podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra. SVM construye un híperplano o conjunto de híperplanos en un espacio de dimensionalidad muy alta (o incluso infinita) que puede ser utilizado en problemas de clasificación o regresión. Una buena separación entre las clases permitirá una clasificación correcta. [11]

2.2.3. Regresion Logística

Es un modelo paramétrico de clasificación binaria dónde la clasificación de los ejemplos se basa en las distancias en la dimensión de un híperplano. [12]

De esos tres, se decidió usar Naive Bayes por varios motivos. Primeramente se descartaron los modelos geométricos debido al desconocimiento de evaluar cómo afectan las características al entrenamiento del modelo. En segundo lugar, se decidió para poder así aplicar los conocimientos adquiridos en la carrera mediante las asignaturas de estadística y Inteligencia Artificial.

Y, finalmente, se decidió usar otro algoritmo basado también en un modelo probabilístico llamado **KNN** [13]. Tanto Naive Bayes como KNN pertenecen a la categoría de algoritmos basados en modelos probabilísticos y, por tanto, parecía interesante compararlos y ver qué resultados se obtenían de cada uno de ellos.

2.2.4. K-Nearest Neighbors (KNN)

El algoritmo KNN también conocido como el algoritmo de los vecinos cercanos consiste en la creación de "n" clústers o también denominadas clases, definidos por el usuario y lo que hace es una clasificación de un nuevo dato en función de la probabilidad de que pertenezca a una de las clases especificadas.

A continuación tenemos un ejemplo de una clasificación knn con 2 clases.

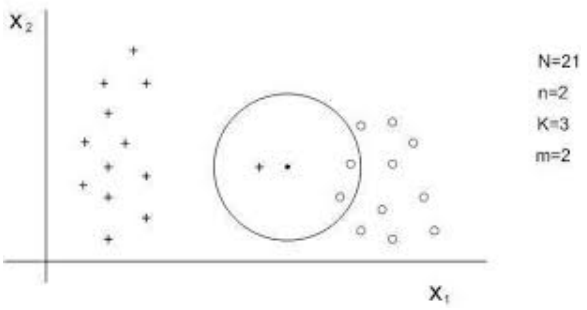

Fig. 8: Ejemplo knn 2 clases

Como se muestra en la figura, en este ejemplo, tenemos 2 clases representadas por diferentes símbolos. Como ya hemos comentado, ante un nuevo ejemplo el comportamiento que seguirá el algoritmo será el de clasificar el ejemplo en una clase en función de la probabilidad de que pertenezca a dicha clase.

3 METODOLOGÍA

Como todo proyecto, se necesitaban tomar unas ciertas decisiones para gestionar el proyecto, en la siguiente sección se van a explicar en una serie de subsecciones, por un lado, cómo se distribuyó el proyecto en cuanto a las tareas y que metodología de desarrollo se utilizó.

Seguidamente pasaremos a explicar con qué lenguaje de programación se desarrolló el tanto el escenario 1, como el primer modelo del escenario 2 así como su justificación. A continuación cómo se implementó el segundo modelo del escenario 2. Y, finalmente que artefactos se generaron para cada uno de los escenarios.

3.1. Metodología de desarrollo

La metodología seguida para distribuir el proyecto en tareas fue mediante Scrum [14]. Por tanto, se planificaron una serie de tareas las cuales se reflejan en los diagramas de Gantt mostrados anteriormente.

Aprovechando las sesiones con el tutor se planificó de tal forma que cada Sprint se cerraba coincidiendo con dichas sesiones. Antes de las reuniones se generaba documentación la cual mostraba en qué estado se encontraba el proyecto, que tareas se habían cumplido hasta ese momento, cuáles no, seguido de una justificación y al final de esas reuniones se decidía que pasos tomar y se re-planificaba si era necesario.

Una vez explicado cómo se estructuró el desarrollo del proyecto el siguiente paso era el de elegir qué lenguaje de programación se utilizaría para la implementación del modelo basado en el algoritmo de Naive Bayes así como también los scripts complementarios que le acompañaban indistintamente del escenario.

3.2. Lenguajes de programación utilizado para la generación del modelo Naive Bayes

Para cualquiera de los escenarios el lenguaje de programación utilizado fue **Python 2.7**. La razón de la elección de este lenguaje fue que ante una primera búsqueda que se realizó, se vio que contenía librerías que nos facilitarían la implementación del algoritmo de entrenamiento. Como segunda razón se decantó la balanza hacia este lenguaje ya que era uno de los que más conocimiento se tenía.

Para la generación del modelo basado en el algoritmo de KNN, no se utilizó Python. Su implementación se detallará a continuación.

3.2.1. Herramienta utilizada para la generación del modelo basado en el algoritmo de KNN

Para el entrenamiento del modelo basado en el algoritmo KNN se utilizó **RapidMiner**[15]. RapidMiner es una herramienta software open-source que permite generar multitud de modelos con diferentes algoritmos de entrenamiento.

Como a todo algoritmo se le introducen unos datos y se define un proceso. Definir un proceso consiste en elegir un algoritmo y configurarlo para que sepa cómo interpretar los datos proporcionados.

Una vez se dispone de los datos y se ha especificado y configurado el proceso simplemente hay que ejecutar y la herramienta sacará unos resultados.

Para evaluarlos simplemente hay que definir un proceso de validación, configurarlo y lanzarlo a ejecutar. El resultado que nos devuelve es una precisión la cual ya puede ser comparada con el modelo de Naive Bayes generado.

Seguidamente, se explicará que artefactos se generaron en cada uno de los escenarios. Como el que se finalizó fue el escenario 2 haremos hincapié en los scripts que se generaron para este pero también comentaremos sin entrar en detalle los que se empezaron a generar para el escenario 1 antes de que se descartara.

3.3. Artefactos generados

A lo largo del documento ya hemos ido mencionando los pasos que se siguieron para la consecución del proyecto, pero ahora vamos a centrarnos en los artefactos que se generaron entrando en detalle en la funcionalidad que implementan cada uno de ellos.

3.3.1. Artefactos generado para el Escenario 2

Para seguir un cierto orden vamos a empezar describiendo el script que se generó para la adaptación de los documentos en el escenario 2.

3.3.2. Dataset Adapter

Este script tomaba como entrada el dataset inicial y leía y almacenaba cada registro del dataset original. Posteriormente se procesaba ese registro dónde primeramente se obtenía la posición a la cual correspondía cada característica, y en esa misma posición se escribiría en un csv que se generaba automáticamente el valor que tenía dicha característica.

Una vez se acababa de procesar y escribir cada característica el último paso que se realizaba era adjuntar la clase. Para ello se realiza una comprobación, si el valor correspondía a -1 automáticamente se transformaba en un 0 y si era 1 se mantenía. Finalmente, después del procesamiento de la clase se adjuntaba después de la última característica.

En el dataset original no había ninguna columna entera de ceros ya que éstas se omitían. Sin embargo, al hacer la adaptación y proporcionar el valor de cada característica en su posición correspondiente podía provocar que se obtuvieran columnas enteras de 0.

Estas columnas no aportaban nada a nuestro entrenamiento, ni ésas, ni cuya desviación estándar equivaliera a 0. Por tanto en este script de adaptación se hacía un proceso el cual se llama **corrección de muestras**. La corrección de muestras consiste en aplicar una pequeña corrección para que nunca exista la probabilidad de alguna de que las probabilidades equivalgan a cero. [16]

Por otro lado se generó el algoritmo de Naive Bayes. A continuación se entrará en detalle en lo que realizaba el algoritmo.

3.3.3. modelNaiveBayes

Éste script contiene todas las funcionalidades necesarias para la implementación del modelo. Inicialmente, en este script parte el dataset en dos subgrupos. El primero es el grupo que forma el set de entrenamiento y el segundo el de test.

Después de haber partido los registros en sets de entrenamiento y test, el set de traning se le separaba los registros por clase. Si era 0 se agrupaba con los registros maliciosos y si era 1 se agrupaba junto a las benignas.

Una vez ya teníamos agrupados los dos tipos de conjuntos se calculaban tanto la desviación estándar como la media de cada conjunto.

Posteriormente, se realizaban tanto las predicciones individuales, como las probabilidades agregadas o condicionales en función de las desviaciones y medias de cada conjunto haciendo así que el modelo aprendiera.

Y, finalmente se calculaba la precisión del modelo en función de las predicciones que había extraído en la fase de entrenamiento y el set de test. [17]

3.3.4. Artefactos generado para el Escenario 1

Por otro lado, para el escenario 1 se generaron algunos otros scripts. Los mencionaremos sin entrar en detalles:

- **Dataset Creator** : Generación de datasets a partir de URLs benignas y maliciosas.
- **UItfg** : Primera versión de la interfaz de la herramienta.

4 RESULTADOS

En esta sección vamos a hablar de los resultados que hemos obtenido a partir de los experimentos realizados con el escenario 2. Expondremos resultados tanto del modelo de Naive Bayes como también una comparación de precisiones con KNN. Además de la comparación de precisiones, también se ha hecho una comparación de tiempos de entrenamiento de los dos algoritmos.

4.1. Resultados del entrenamiento del modelo de Naive Bayes

A partir de los scripts que generamos obtuvimos un modelo de entrenado mediante unos datos reales con una precisión de acierto máximo de un **noventa y cinco por ciento** de las veces.

La siguiente tabla muestra los valores obtenidos incrementando el número de features en los datasets, entrenando el modelo con éstos y testeándolo con los subconjuntos de URLs de test generados para cada iteración y con los cuales no ha sido entrenado.

La manera de obtener la máxima precisión fue de manera exhaustiva. Esto significa que se hicieron numerosas pruebas cada vez aumentando más el número de features y analizando las precisiones obtenidas.

En una primera apreciación de la tabla 1 se puede ver como inicialmente con pocas features la precisión es relativamente baja rondando el 70 %. A medida que el número de features crece la precisión va oscilando hasta un cierto punto en la que el modelo empieza a mantenerse estable.

Esto se debe a que en todo modelo del Aprendizaje Automático inicialmente la precisión del entrenamiento entrenado con más datos crece exponencialmente y llega un punto en la que el crecimiento de la precisión se atenúa y entra

TAULA 1: TABLA DE PRECISIÓN-ACCURACY

#Características	Precisión(%)
5	70
100	86
200	91
300	93
400	92
500	92
1000	82
1500	83
2000	83
4000	83
6000	82
8000	83

en un punto de estabilidad. Más adelante se mostrará un gráfico dónde se podrá apreciar ésta curva de aprendizaje.

Pero antes de eso, nos centraremos un poco más en ése crecimiento exponencial. En la siguiente tabla iremos algo más despacio incrementando el número de features para que se pueda apreciar como contra más features más mejora el modelo teniendo en cuenta de que éstas features **aportan al modelo**. Aumentar el número de features no es indicativo de aumentar la precisión. Si aumentamos las features pero dichas características no son buenas e introducen ruido al modelo, éste no mejorará sino al contrario, empeorará.

TAULA 2: TABLA DE PRECISIÓN-ACCURACY DE 5 ENTRE 35 FEATURES

#Características	Precisión(%)
2	55
4	72
5	70
6	91
8	90

Como se puede apreciar en esta tabla, el modelo empieza con una precisión de 55 % y a partir de 8 características ya empieza a alcanzar las máximas precisiones. El hecho de que con 8 features el modelo alcance una buena precisión puede deberse a que esas primeras 8 características son de una gran calidad para el modelo y probablemente aporten más que otras.

A continuación, en la siguiente figura se puede ver la gráfica de la curva de aprendizaje seguida por el modelo al incrementar el número de features.

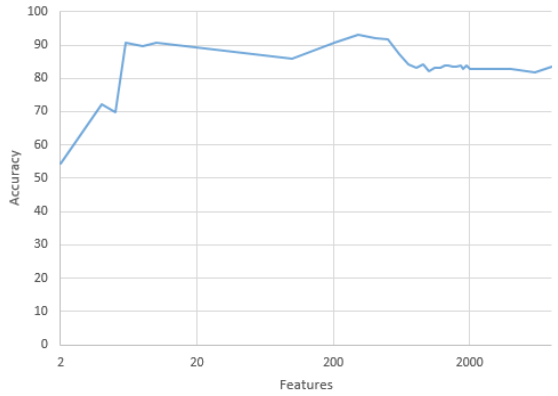

Fig. 9: Curva de aprendizaje del modelo Naive Bayes

En esta figura puede verse lo que se decía, al inicio la curva crece muy rápidamente pero, a medida que incrementan las características el modelo entra en un estado de estabilidad alrededor del 83 %.

Esta gráfica es perfecta para mostrar lo que se decía anteriormente y demostrar uno de los objetivos esmentados. La calidad de los datos con los que se entrena el modelo es muy importante, sino puede darse el caso de que aunque aumentemos los datos el modelo no mejore. Aun así, si incrementáramos el número de features y dichas features fueran de calidad el modelo seguiría mejorando la precisión.

Finalmente, en éste proyecto se ha llegado a entrenar el modelo con un máximo de 8000 features de las 3.2 millones de las que se comentaban. Lamentablemente, debido a la falta de potencia de cómputo no se han podido realizar pruebas con más features.

4.2. Comparación de precisiones y tiempo entre Naive Bayes y KNN

Después de entrenar el modelo usando Naive Bayes se decidió comparar con otro algoritmo también probabilístico. Para ello se usó RapidMiner y se entrenó un modelo con los mismos datos que se usaron para entrenar a Naive Bayes. A continuación en la siguiente figura se muestra un gráfico de la comparación de los dos modelos.

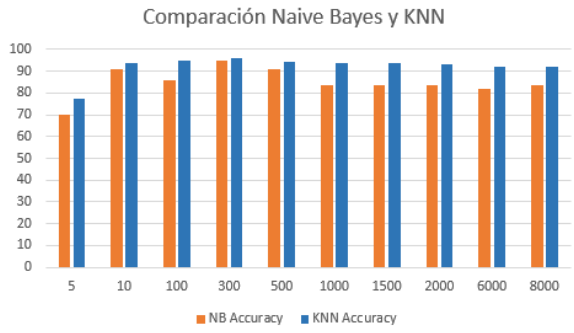

Fig. 10: Gráfico de comparación Naive Bayes y KNN

Como se puede apreciar en el siguiente gráfico el KNN consigue unos mejores resultados en cuanto a precisión que el Naive Bayes. Este resultado lo único que nos indica es que para este caso en concreto y con éstos datos el algo-

ritmo de clasificación de K-vecinos cercanos funciona más eficientemente que el Naive Bayes.

Aun así, en éste gráfico puede apreciarse lo que se comentaba anteriormente. Sin importar cuál de los dos dé mejores resultados, los dos empiezan con precisiones de acierto bajas y a medida que se van añadiendo características a su entrenamiento éstos se van obteniendo mejores precisiones.

Tampoco es de extrañar que en algunos momentos la precisión baje. Como comentábamos anteriormente, puede ser que unos ciertos datos añadan ruido al modelo y aporten mucho menos que otros. Si conociésemos a qué corresponde cada una de las features de los datasets podríamos haber planteado una estrategia para quedarnos con aquellos que aportaran mucho al modelo maximizando así la precisión.

Otro factor interesante a comparar era el tiempo que necesitaban los algoritmos para entrenarse. La virtud del Naive Bayes es que es un algoritmo fácil de utilizar y rápido. Sin embargo el KNN no es tan rápido como el Bayes aunque también es sencillo de entender pues su base matemática se basa en cálculos estadísticos.

Veamos una tabla de comparación de tiempos necesarios para el entrenamiento de los dos algoritmos. La tabla es la que se muestra a continuación:

Comparación features - Tiempo		
	Features	**Tiempo (min)**
Naive Bayes	1000	1
	2000	2
	6000	7
	8000	10
KNN	1000	19
	2000	33
	6000	90
	8000	180

Como se puede apreciar en esta tabla, se demuestra lo que comentábamos anteriormente. Para entrenar unas dos mil features el algoritmo de Naive Bayes tarda aproximadamente unos dos minutos tanto como para entrenarse como para testearse.

Sin embargo, el KNN para unas dos mil features necesita un tiempo aproximado de 33 minutos. Esto es debido a que en realidad no se necesita tanto tiempo para entrenarse pero sí que a la hora de testearlo es más costoso que con Bayes. Para el caso con más features se puede ver que la diferencia es abismal. Mientras que Bayes necesita solo de unos 10 minutos, el KNN toma aproximadamente unas 3 horas.

Naturalmente, contra más sean las features, más incrementan los datos de entrenamiento y así lo hace también el tiempo que se necesita para realizar el entrenamiento.

En resumen, por un lado tenemos que, aunque KNN ofrece mejores resultados, el tiempo de entrenamiento de éste se ve claramente superado por Naive Bayes.

5 CONCLUSIONES

Motivado por el gran número de ciberataques que se producen a lo largo del día en cualquier parte del mundo y el atractivo de la nueva rama de Inteligencia Artificial, el objetivo inicial de éste proyecto era la de explorar técnicas del Aprendizaje Automático y realizar una herramienta que fuera capaz de predecir un ataque malicioso realizado mediante el acceso de una URL maligna.

Durante el transcurso del proyecto y debido a situaciones, tanto los objetivos como el enfoque del proyecto han ido cambiando. El objetivo del proyecto pasó de ser el de la creación de la herramienta al de explorar técnicas del Aprendizaje Automático para generar un modelo entrenado mediante datos reales para predecir si una URL es maliciosa o no.

A su vez, y debido a las mismas situaciones, el enfoque del primer escenario pasó a ser el de la extracción de características y, por tanto se descartó ya que lo que se buscaba y el enfoque que se quería dar era el de ante la premisa de un cierto tipo de ataque, utilizar técnicas de Inteligencia Artificial que predijeran que un ataque se podría producir.

Estos cambios de objetivos y enfoques supusieron unos cambios en la planificación, la generación de artefactos diferentes pero que gracias a una buena gestión del cambio se supieron manejar y adaptar el tiempo del que se disponía para la correcta finalización del proyecto. Por supuesto hubieron una serie de limitaciones como podría ser la falta de cómputo con la que si se hubiera dispuesto quizás se podrían haber obtenido resultados y precisiones más altas.

Este proyecto ha sido realmente útil para introducirse en un mundo en constante expansión como es el de la Inteligencia Artificial y en concreto del Aprendizaje Automático. Se ha entendido que hay diferentes tipos de modelos dentro del Aprendizaje Automático y se ha optado por la utilización de los modelos probabilísticos.

Dentro de los modelos probabilísticos se ha indagado en los algoritmos de clasificación supervisada los cuales eran Naive Bayes y K-vecinos más cercanos. Se ha visto que implicaciones tiene usar Naive Bayes, cómo solucionar las deficiencias que tiene, comprobar cómo afectan los datos que se les inyectan a los modelos.

Se ha comprobado que, a la hora de entrenar, no importa la cantidad de datos si dichos datos no son de calidad. Además, se ha descubierto que había características que aportaban más al modelo que otras.

Se podría decir que éste proyecto ha servido para sumergirnos en un mundo el cual aún tiene mucho que decir y hacer. Cada vez más se busca que los programas sean capaces de aprender y conseguir que sean cada vez más autónomos. Por ello se considera que en ese sentido ha sido muy útil ya que de cara al futuro se tendrá una base sólida con la que investigar al respecto.

AGRADECIMIENTOS

El presente trabajo de investigación fue realizado bajo la supervisión de mi tutor Jordi Duran Cals a quién me gustaría expresar mi más profundo agradecimiento, por hacer posible la realización de este estudio. Además, de agradecer su paciencia, tiempo y dedicación que tuvo para que esto saliera de manera exitosa.

A mi compañero de trabajo y amigo Joan Salvatella Ibáñez pues su disposición, sus ganas de ayudar, sus aportes, sus consejos y su paciencia me han ayudado mucho a la realización de éste proyecto. Agradecer de todo corazón que se haya ofrecido a ayudarme cuando no tenía la necesidad de hacerlo y aún así se mostró tan amable conmigo. Muchas gracias.

REFERENCIAS

[1] "Internet en el mundo," 2016. [Accedida el 04 Junio 2016].

[2] "Cómo son los diez ciberataques más frecuentes en internet," 2016. [Accedida el 22 Junio 2016].

[3] "Blacklist (computing)," 2016. [Accedida el 24 Junio 2016].

[4] S. S. G. M. V. Justin MA, Lawrence K. Saul, "Identifying suspicious urls: An application of large-scale online learning," p. 2.

[5] "Detecting malicious url," 2016. [Accedida el 18 Junio 2016].

[6] "Svm-light," 2016. [Accedida el 11 Junio 2016].

[7] "Inteligencia artificial," 2016. [Accedida el 04 Junio 2016].

[8] "Arboles de decisión," 2016. [Accedida el 04 Junio 2016].

[9] "Aplicaciones del machine learning," 2016. [Accedida el 18 Junio 2016].

[10] "Naive bayes," 2016. [Accedida el 12 Junio 2016].

[11] "Support vector machine," 2016. [Accedida el 12 Junio 2016].

[12] S. S. G. M. V. Justin MA, Lawrence K. Saul, "Beyond blacklists: Learning to detect malicious web sites from suspicious urls," p. 3.

[13] "k-nearest neighbors algorithm," 2016. [Accedida el 22 Junio 2016].

[14] "Qué es scrum," 2016. [Accedida el 22 Junio 2016].

[15] "Rapidminer," 2016. [Accedida el 22 Junio 2016].

[16] "Naive bayes ingenuo," 2016. [Accedida el 12 Junio 2016].

[17] "How to implement naive bayes from scratch in python," 2016. [Accedida el 20 Junio 2016].