MEMORIA PRÁCTICA 2 SISTEMAS DE INFORMACIÓN

Marcos Bonilla Cubero [m.bonillac.2018@alumnos.urjc.es](mailto:m.bonillac.2018@alumnos.urjc.es) 70068909F

Pablo Gracia Correa [p.gracia.2019@alumnos.urjc.es](mailto:p.gracia.2019@alumnos.urjc.es) 50257261F

Para la realización de esta práctica hemos usado la librería de Flask y además hemos creado una plantilla html para desde ahí poder dirigirnos a las diferentes partes de la práctica.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Ejercicio 2**

**El top X de usuarios críticos**

Entendemos que por top X de usuarios se refiere a un número “x” de usuarios. Pedimos por pantalla el tamaño del top y lo pasamos por una petición POST a esta ruta:

Texto

Descripción generada automáticamente

Nos llega el tamaño del top y lo pasamos a la función criticUsers muy parecida a otra que teníamos en la otra parte de la práctica, solo que en vez de devolver siempre 10 usuarios nos devuelve los que le pasemos.

Texto

Descripción generada automáticamente

Ordenamos los usuarios por nivel de criticidad que lo calculamos con emails clicados entre el total de emails, y devolvemos el top que se nos pide.

Después los mostramos con un template:

Texto, Carta

Descripción generada automáticamente

**El top X de páginas web vulnerables**

De manera muy similar al anterior ejercicio pedimos al usuario que introduzca un número por el CMI para mostrar una cantidad de usuarios en el top.

Texto

Descripción generada automáticamente

Al igual que antes modificamos una de las funciones de la otra práctica para que nos devuelva el número de webs que le pasemos. Lo que hace la función outdatedWebs para ver que webs son más vulnerables es ver que webs están más desactualizadas respecto a políticas:

Texto

Descripción generada automáticamente

Y las mostramos en un template:

Texto, Carta

Descripción generada automáticamente

**Ejercicio 3**

Ahora tendremos que hacer que se pueda ver los usuarios que tienen más de un 50% de criticidad.

Como se puede ver en la página principal del CMI que mostramos al principio de la memoria hemos añadido unos botones para que se puedan elegir si mostrar los que tienen más del 50% o menos del 50%.

Y se pasa a la misma ruta y función que el anterior ejercicio, a topXUsuariosVulnerables. Se puede ver en esa función que obtenemos los valores de los botones para saber si se han pulsado o no. En caso de que se haya pulsado alguno o los dos se llama a la función usersSpam para que nos devuelva una lista con los usuarios con más o menos del 50% de emails de Spam/Phising clicados (criticidad):

Texto

Descripción generada automáticamente

Después lo mostramos en el mismo template que el del top de usuarios en el que tenemos una serie de ifs para que se muestre o no:

Texto

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Ejercicio 4**

En este ejercicio tendremos que mostrar las últimas 10 vulnerabilidades en tiempo real, para ello tendremos que hacer web scrapping. Vemos en la página <https://www.cve-search.org/api/> que hay una api para poder obtener un JSON con las últimas 30 vulnerabilidades.

Para ello hacemos una petición GET a la página <https://cve.circl.lu/api/last> y lo parseamos para obtener los últimos cve-id y mostrarlos.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**Ejercicio 5**

Para este ejercicio hemos decidido hacer un sistema de login y para ello usaremos la librería de Flask\_login.

Para ello nos creamos las clases User y ModelUser:

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente Texto

Descripción generada automáticamente

La clase User la creamos con tan solo id y contraseña ya que no nos es necesario complicarla más.

Inicializamos el Login Manager:



Y establecemos una super secret key:



Para que el sistema de login funcione tenemos que crear la función de loadUser

Texto

Descripción generada automáticamente

La función en la que se comprueba si el usuario está en la base de datos es ésta:

Texto

Descripción generada automáticamente

Si la petición es un GET devuelve un template para que el usuario meta usuario y contraseña, si es un POST comprueba que el nombre y contraseña coinciden con los de la base de datos. La contraseña al estar hasheada con md5 en la base de datos, la contraseña que se introduce tenemos que hashearla para comprobar que es correcta.

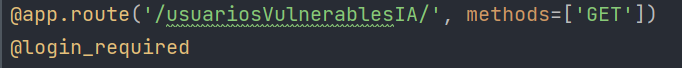
También hemos hecho un logout:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Ya hay una función de desloguearse en flask\_login así que solo la llamamos.

Y por último tenemos que establecer para que rutas es obligatorio loguearse para acceder, lo hacemos poniendo @login\_required en las rutas:



Establecemos el login obligatorio en todas las rutas menos en que nos devuelve las últimas vulnerabilidades ya que es algo que puede consultar cualquiera.

Además, creamos una página muy sencilla que salte cuando hay error de que el usuario no está autorizado a entrar a la página:

Texto

Descripción generada automáticamente

**Ejercicio 6**

En este ejercicio tendremos que implementar 3 métodos de machine learning, regresión lineal, decisión tree y random forest.

Lo primero que hacemos es dos funciones en la que parseamos los users, en una con la criticidad ya calculada y otra sin calcular:

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

**Regresión Lineal**

Texto

Descripción generada automáticamenteEn esta función usando la librería de sklearn implementamos el método de regresión lineal, primero entrenamos los datos con los usuarios del JSON con las clases(vulnerable, no vulnerables) y luego hacemos la predicción. Al darnos valores decimales aproximamos, todo lo que esté por debajo de 0.5 es 0 y viceversa.

Después lo mostramos usando matplotlib:

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

**Decision Tree**

El siguiente algoritmo que implementamos es el de Decision Tree:

Captura de pantalla de computadora

Descripción generada automáticamente

Simplemente usamos la función de DecionTreeClassifier del sklearn, primero la entrenamos u luego lo predecimos. Este algoritmo se basa en ir tomando decisiones en cuanto a los datos que tenemos para llegar a una conclusión, el árbol de decisión que nos queda es este:

Imagen de la pantalla de un celular con texto e imágenes

Descripción generada automáticamente con confianza media

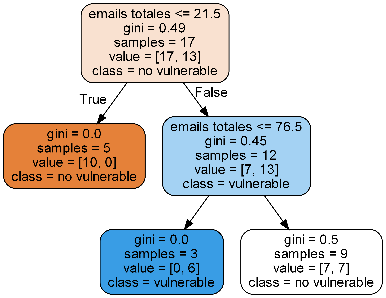
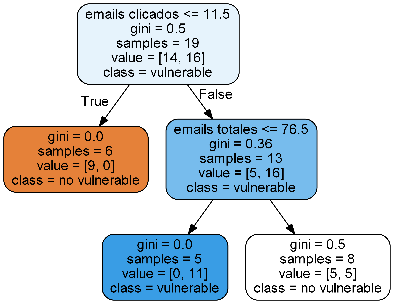
**Random Forest**

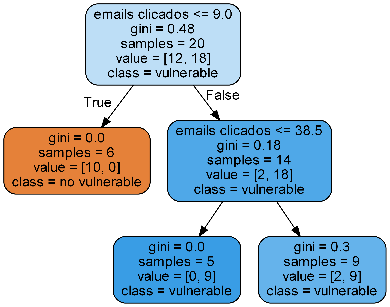
Este algoritmo es similar al de Decision Tree por el hecho de que también se basa en decisiones, pero la diferencia es que en el de Decision Tree solo tenemos un árbol de decisión y en este tenemos bastantes, y las decisiones se basan teniendo en cuenta todos.

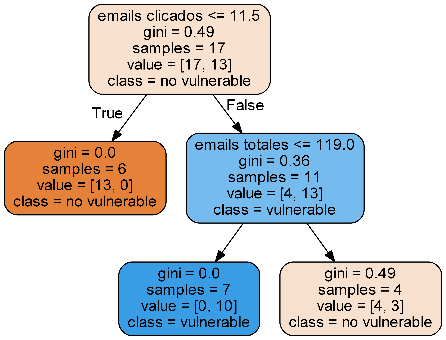
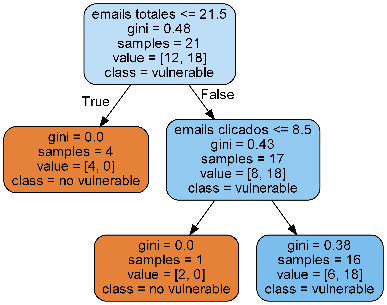
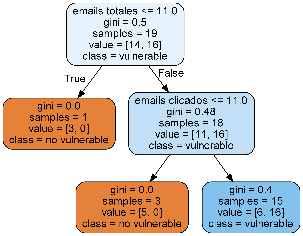
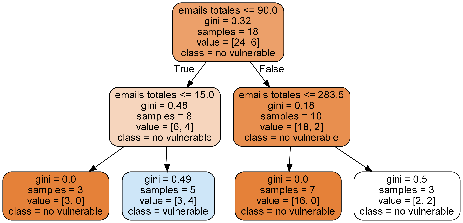
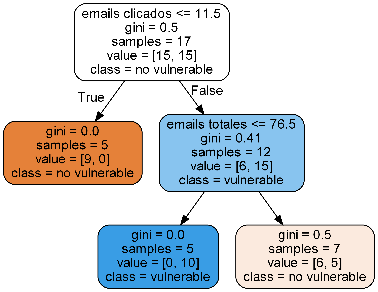
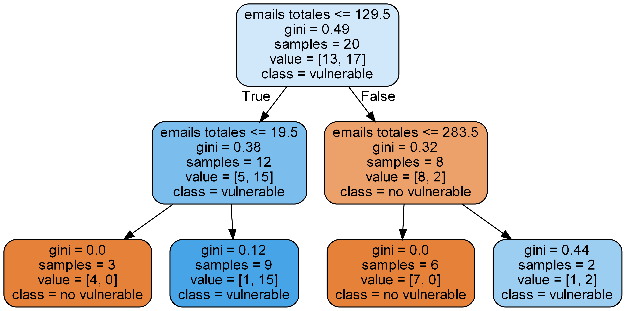
Texto

Descripción generada automáticamente

Al igual que en los otros algoritmos usamos una librería de sklearn, primero entrenamos y luego predecimos.

Diagrama

Descripción generada automáticamenteSe generan 10 árboles:



Para mostrar los resultados nos vamos a esta ruta y mostramos cada usuario si es vulnerable o no en la predicción en cada uno de los 3 métodos:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamenteY así con los 3 algoritmos.

**GitHub**

<https://github.com/pgraciac/PracticaSI>