

Universidad
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE LA CIBERSEGURIDAD

Práctica 2: Representación de los datos

METODOLOGÍAS DE DESARROLLO SEGURO

CURSO 2021-2022

[LINK GITHUB](#)

**Redactado por: Alejandro Gallego Sánchez
Adriano Campos Soto
Ismael Gómez Esquilichi**

Índice

1. Gráficos dinámicos	1
1.1. Top X usuarios críticos	1
1.2. Top X páginas webs vulnerables	2
2. Top X usuarios críticos filtrado	3
3. Mostrar las últimas 10 vulnerabilidades a tiempo real	3
4. Nueva funcionalidad	4
4.1. Sistema de login para usuarios	4
4.2. API <i>exploitdb</i>	6
4.3. Modelo de predicción usuario crítico	7
5. Machine Learning	8
5.1. Decision Tree	8
5.2. Random Forest	9
5.3. Regresión Linear	11

1. Gráficos dinámicos

Durante el desarrollo de la práctica se nos solicitaba cambiar los gráficos generados anteriormente de manera estática, por gráficos que se generasen de manera dinámica. Para ello hemos decidido utilizar la librería *plotly* para Python3. Esta librería permite generar gráficos dinámicos a partir de los *dataframes* de la librería de *pandas* para Python3 también. Además se han añadido una serie de modificadores que permitiesen establecer transformar los parámetros para obtener exactamente los gráficos que necesitábamos.

1.1. Top X usuarios críticos

Durante este apartado de la práctica, se nos solicitaba el gráfico que obteníamos anteriormente el cual mostraba los usuarios más vulnerables de la página web. Este gráfico se genera apartir del siguiente método:

```
c, conn = connect_db("Entrega1/database/database.db")
df1 = pd.read_sql_query("select * from users", conn)
df1 = porcentaje_peligro(df1)
df1 = usuarios_criticos(df1).head(x)
data = plotlyF(df1, x)

def plotlyF(df: pd.DataFrame, x):
    trace = go.Bar(x=df["username"], y=df["porcentaje_click"])
    layout = go.Layout(title="Top %s usuarios críticos" % (x), xaxis=dict(title="us",
                                                                    yaxis=dict(title="porcentaje_click"))
    data = [trace]
    fig = go.Figure(data=data, layout=layout)
    fig_json = json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)
    return fig_json
```

Estas funciones permiten generar el gráfico que después mostramos a traves del template de *jinja2*:

```
{% if data %}
<div id='chart' class='chart'></div>
<script src='https://cdn.plot.ly/plotly-latest.min.js'></script>
<script type='text/javascript'>
    var graphs = {{data | safe}};
    Plotly.plot('chart',graphs,{});
</script>
{% endif %}
```

Esto nos permite obtener el siguiente resultado:

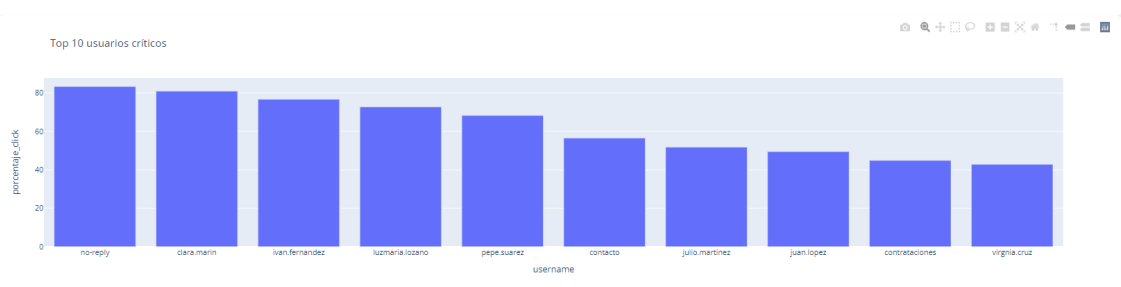


Figura 1: Gráfico de usuarios generado con *plotly*

1.2. Top X páginas webs vulnerables

Este apartado es similar al anterior, con la salvedad de que esta vez se nos solicitaban sobre las webs vulnerables. Siguiendo con el procedimiento de *plotly* hemos generado el gráfico de la siguiente manera:

```
c, conn = connect_db("Entrega1/database/database.db")
dframe2 = pd.read_sql_query("SELECT * FROM legal", conn)
dframe2 = get_paginas_desactualizadas(dframe2, limit=pages)
print(dframe2)
data2 = plotlyP(dframe2, pages)

def plotlyP(df: pd.DataFrame, x):
    trace = go.Bar(x=df["url1"], y=df["n_politicas"])
    layout = go.Layout(title="Top %s webs vulnerables" % (x), xaxis=dict(title="web",
                                                                    yaxis=dict(title="porcentaje_click"))
    data = [trace]
    fig = go.Figure(data=data, layout=layout)
    fig_json = json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)
    return fig_json
```

Lo que insertado en la *template* de *jinja2* quedaría de la siguiente manera:

```
{% if data2 %}
<div id='chart2' class='chart'></div>
<script src='https://cdn.plot.ly/plotly-latest.min.js'></script>
<script type='text/javascript'>
    var graphs2 = {{data2 | safe}};
    Plotly.plot('chart2',graphs2,{});
</script>
{% endif %}
```

Obteniendo el siguiente gráfico como resultado:

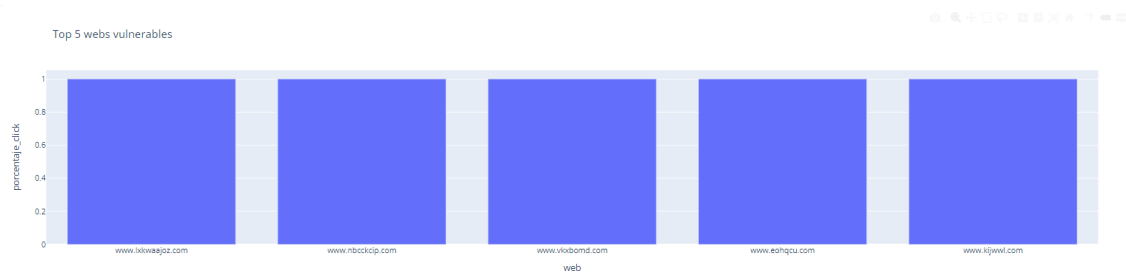


Figura 2: Gráfico de webs generado con *plotly*

2. Top X usuarios críticos filtrado

Este apartado está relacionado directamente con el anterior. Tendremos que tratar los *dataframes* utilizados en el apartado 1.1 para poder obtener los valores que necesitamos. Para ello añadiremos otra función a nuestro servicio:

```
def tratar_dataframe(df: pd.DataFrame, b: bool):
    if not b:
        df = df.loc[df['porcentaje_click'] < 50]
    else:
        df = df.loc[df['porcentaje_click'] >= 50]
    return df
```

A la cual habrá que llamar antes de llamar a la función *plotlyF()* de la siguiente manera:

```
c, conn = connect_db("Entrega1/database/database.db")
df1 = pd.read_sql_query("select * from users", conn)
df1 = porcentaje_peligro(df1)
df1 = usuarios_criticos(df1)
df1 = tratar_dataframe(df1, critico)
data = plotlyF(df1.head(x), x)
```

Los parámetros para generar los nuevos gráficos se pasan como variables *GET* en la petición.

3. Mostrar las últimas 10 vulnerabilidades a tiempo real

Para el desarrollo de este apartado hemos usado la API de <https://cve.circl.lu/api/last> la cual nos permite obtener una lista con los últimos CVEs reportados y publicados. Hemos desarrollado una función que a través de una petición *GET* recogiese todas las vulnerabilidades y hemos formateado la respuesta para poder generar una lista con lo mismo. La función utilizada ha sido:

```
def ejercicio4():
    r = requests.get("https://cve.circl.lu/api/last")
    jotason = json.loads(r.text)
```

```

1 = []
for i in range(10):
    temp = jotason[i]
    l.append(temp)
return l

```

Luego hemos utilizado una librería de javascript, *chartjs*, para poder mostrar los resultados devueltos en formato *json* con un formato más amigable para el usuario. El resultado es el siguiente:

ID	CVSS	Resumen
CVE-2021-3596	4.3	A NULL pointer dereference flaw was found in ImageMagick in versions prior to 7.0.10-31 in ReadSVGImage() in coders/svg.c. This issue is due to not checking the return value from libxml2's xmlCreatePushParserCtxt() and uses the value directly, which leads to a crash and segmentation fault.
CVE-2021-36740	6.4	Varnish Cache, with HTTP/2 enabled, allows request smuggling and VCL authorization bypass via a large Content-Length header for a POST request. This affects Varnish Enterprise 6.0.x before 6.0.8r3, and Varnish Cache 5.x and 6.x before 6.5.2, 6.6.x before 6.6.1, and 6.0 LTS before 6.0.8.
CVE-2021-41965	None	A SQL injection vulnerability exists in ChurchCRM version 2.0.0 to 4.4.5 that allows an authenticated attacker to issue an arbitrary SQL command to the database through the unsanitized EN_tyid, theID and EID fields used when an Edit action on an existing record is being performed.
CVE-2021-42072	6.5	An issue was discovered in Barrier before 2.4.0. The barriers component (aka the server-side implementation of Barrier) does not sufficiently verify the identity of connecting clients. Clients can thus exploit weaknesses in the provided protocol to cause denial-of-service or stage further attacks that could lead to information leaks or integrity corruption.
CVE-2022-1292	10.0	The c_rehash script does not properly sanitise shell metacharacters to prevent command injection. This script is distributed by some operating systems in a manner where it is automatically executed. On such operating systems, an attacker could execute arbitrary commands with the privileges of the script. Use of the c_rehash script is considered obsolete and should be replaced by the OpenSSL rehash command line tool. Fixed in OpenSSL 3.0.3 (Affected 3.0.0.3.0.1.3.0.2). Fixed in OpenSSL 1.1.1o (Affected 1.1.1-1.1.1n). Fixed in OpenSSL 1.0.2ze (Affected 1.0.2-1.0.2zd).
CVE-2022-21426	5.0	Vulnerability in the Oracle Java SE, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: JAXP). Supported versions that are affected are Oracle Java SE: 7u331, 8u321, 11.0.14, 17.0.2, 18; Oracle GraalVM Enterprise Edition: 20.3.5, 21.3.1 and 22.0.0.2. Easily exploitable vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized ability to cause a partial denial of service (partial DOS) of Oracle Java SE, Oracle GraalVM Enterprise Edition. Note: This vulnerability applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. This vulnerability can also be exploited by using APIs in the specified Component, e.g., through a web service which supplies data to the APIs. CVSS 3.1 Base Score 5.3 (Availability impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/CN:I/L:N/A). Vulnerability in the Oracle Java SE, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: Libraries). Supported versions that are affected are Oracle Java SE: 7u331, 8u321, 11.0.14, 17.0.2, 18; Oracle GraalVM Enterprise Edition: 20.3.5, 21.3.1 and 22.0.0.2. Easily exploitable vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized update, insert or delete access to some of Oracle Java SE, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. This vulnerability can also be exploited by using APIs in the specified Component, e.g., through a web service which supplies data to the APIs. CVSS 3.1 Base Score 5.3 (Integrity impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/CN:I/L:N/A).
CVE-2022-21434	5.0	Vulnerability in the Oracle Java SE, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: JNDI). Supported versions that are affected are Oracle Java SE: 7u331, 8u321, 11.0.14, 17.0.2, 18; Oracle GraalVM Enterprise Edition: 20.3.5, 21.3.1 and 22.0.0.2. Easily exploitable vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized update, insert or delete access to some of Oracle Java SE, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. This vulnerability can also be exploited by using APIs in the specified Component, e.g., through a web service which supplies data to the APIs. CVSS 3.1 Base Score 5.3 (Integrity impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/CN:I/L:N/A).
CVE-2022-21496	5.0	Vulnerability in the Oracle Java SE, Oracle GraalVM Enterprise Edition product of Oracle Java SE (component: JNDI). Supported versions that are affected are Oracle Java SE: 7u331, 8u321, 11.0.14, 17.0.2, 18; Oracle GraalVM Enterprise Edition: 20.3.5, 21.3.1 and 22.0.0.2. Easily exploitable vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized update, insert or delete access to some of Oracle Java SE, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. This vulnerability can also be exploited by using APIs in the specified Component, e.g., through a web service which supplies data to the APIs. CVSS 3.1 Base Score 5.3 (Integrity impacts). CVSS Vector: (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/CN:I/L:N/A).
CVE-2022-28463	None	ImageMagick 7.1.0-27 is vulnerable to Buffer Overflow.
CVE-2022-30708	None	Webmin through 1.991, when the Authentic theme is used, allows remote code execution when a user has been manually created (i.e., not created in Virtualmin or Cloudmin). This occurs because settings-editor_write.cgi does not properly restrict the file parameter.

Figura 3: Lista con los 10 últimos CVE

4. Nueva funcionalidad

Como parte de la práctica se solicitaba implementar nueva funcionalidad al CMI, las opciones elegidas para este apartado han sido el sistema de login de usuarios, conexión con otra API en este caso *exploitdb* y un modelo de *machine learning* para poder predecir si un usuario será vulnerable o no.

4.1. Sistema de login para usuarios

Para implementar este modelo se ha tomado la decisión de continuar con el modelo de base de datos utilizado para la generación de los *dataframes*, se ha utilizado una base de datos *sqlite* y las funciones de Flask para poder crear sesiones. Se han implementado las funciones básicas de registro, login y logout. Además se ha impuesto que para poder acceder al dashboard el usuario tendría que estar *logueado* dentro de la propia aplicación. También se han añadido los endpoints y el html necesario para poder generar ambas vistas. Las funciones propias de este apartado son:

```

@app.route('/register', methods=['GET', 'POST'])
def register_page():
    if request.method == 'GET':
        return render_template('pages/sign-up.html')
    elif request.method == 'POST':
        user = request.form['username']
        password = request.form['password']
        try:
            register_user(user, password)
            return redirect(url_for('login_page'))
        except sqlite3.IntegrityError as e:
            return render_template('pages/sign-up.html', error="Ya existe ese nombre de usuario")
        except Exception as e:
            print(e)
            return redirect(url_for('register_page'))

@app.route('/login', methods=['GET', 'POST'])
def login_page():
    if request.method == 'GET':
        return render_template('pages/sign-in.html')
    else:
        user = request.form['user']
        passwd = request.form['password']
        log = login_user(user, passwd)
        if log == True:
            session['user'] = (user)
            return redirect(url_for('dashboard'))
        else:
            return render_template('pages/sign-in.html', error="Credenciales no validos, registrate")

@app.route('/logout')
def log_out():
    if 'user' in session:
        session.pop('user')
    return redirect(url_for('login_page'))

```

Y respecto a las funciones propias de *SQL* las mismas serían:

```

def register_user(username: str, password: str):
    try:
        cursor.execute("""INSERT INTO users_flaskitos(username, passwordHash)
            VALUES (?, ?)""",

```

```

                                (sqlescape(username), sqlescape(hash_password(password))))
except sqlite3.IntegrityError as pr:
    raise
con.commit()

def login_user(username: str, password: str):
    com = cursor.execute("""SELECT username, passwordHash FROM users_flaskitos
        WHERE username=?""",
                        [sqlescape(username)])
    res = com.fetchall()
    if len(res) == 1:
        if res[0][1] == hash_password(password):
            return True
        else:
            return False
    elif len(res) == 0:
        return "No existe" # Cambiar a excepcion
    else:
        return "No te portes mal"

```

Esto nos permite desarrollar toda la lógica de las sesiones de usuario mediante la propiedad *session* de Flask, esta sólo necesita añadirse y se encarga de gestionar los tokens necesarios en el cliente. Para autenticar a un cliente sólo necesitamos esta línea:

```
session['user'] = (user)
```

4.2. API *exploitdb*

Como segunda funcionalidad hemos decidido implementar una lista que contuviese los últimos exploits publicados en la famosa plataforma *exploitdb*. Para ello hemos utilizado su propia API, la cual incluye un endpoint que devuelve los datos en formato XML <https://www.exploit-db.com/rss.xml>. Para recoger los resultados y poder tramitarlos a nuestra librería de javascript hemos implementado la siguiente función:

```

def exploitdb():
    exploitdb_list = []
    last_exploitdb = feedparser.parse("https://www.exploit-db.com/rss.xml")
    for article in last_exploitdb.entries:
        name = article.description
        link = article.link
        category = article.title.split(" ")[0][1:]
        exploitdb_list.append({"name": name, "category": category,
                               "link": link, "saved": False})
    return exploitdb_list

```

Y de nuevo hemos usado *chartjs* para mostrarlo en nuestro HTML, obteniendo este resultado:

Últimas entradas de ExploitDB

Show 10 entries

Descripción del Exploit	Enlace	Categoría
Akka HTTP 10.1.14 - Denial of Service	link	remote
Anuko Time Tracker - SQLi (Authenticated)	link	webapps
Apache CouchDB 3.2.1 - Remote Code Execution (RCE)	link	remote
Beehive Forum - Account Takeover	link	webapps
Bitrix24 - Remote Code Execution (RCE) (Authenticated)	link	webapps
Bookken Notes - Directory Traversal	link	remote
College Management System 1.0 - 'course_code' SQL Injection (Authenticated)	link	webapps
CS2 CMS 1.3.0 - 'Multiple' Blind SQLi	link	webapps
Cyclos 4.14.7 - 'groupid' DOM Based Cross-Site Scripting (XSS)	link	webapps
Cyclos 4.14.7 - DOM Based Cross-Site Scripting (XSS)	link	webapps

Showing 1 to 10 of 50 entries

Previous 1 2 3 4 5 Next

Figura 4: Lista con los últimos exploits de *exploitdb*

4.3. Modelo de predicción usuario crítico

Hemos implementado un método en *Flask* que recibe como parámetros el nombre de un usuarios, los correos de phishing que ha recibido y el número de ellos en los que ha hecho click.

Para construir el modelo, hemos usado los datos dados para hacer el apartado de *machine learning*, en concreto hemos utilizado el algoritmo *Random Forest* para el clasificador. Lo generamos con el siguiente código.

```
@app.route('/iatraining', methods=['POST'])
def train_ai():
    if request.method == 'POST':
        name = request.form['nombre']
        erecibidos = int(request.form['erecibidos'])
        eclickados = int(request.form['eclickados'])
        if eclickados > erecibidos:
            return 'Error'
        x_train = datos.drop('vulnerable', axis=1)
        y_train = datos['vulnerable']

        rf_clf = RandomForestClassifier(max_depth=2,
                                       random_state=0, n_estimators=10)
        rf_clf.fit(x_train, y_train)

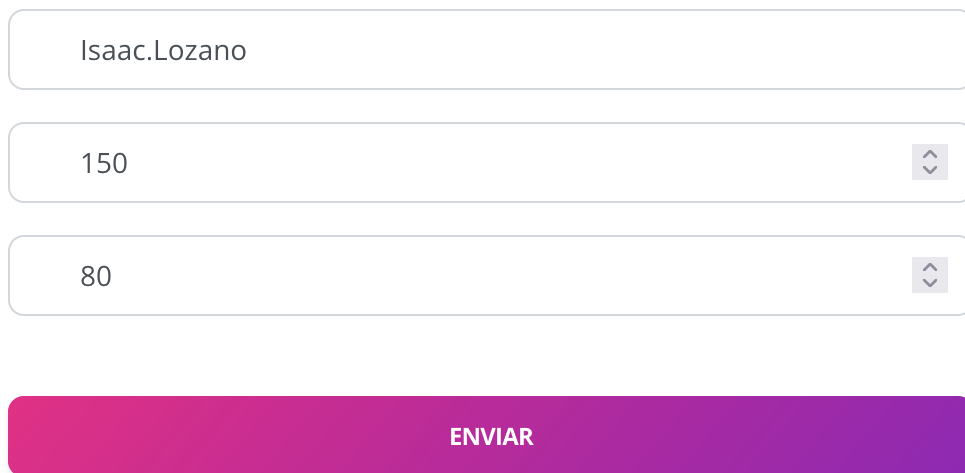
        predecir = pd.DataFrame({'emails_phishing_recibidos':
                                [erecibidos], 'emails_phishing_clicados': [eclickados]})
        # Predict data on test part
        y_predict = rf_clf.predict(predecir)
        if y_predict[0] == 1:
            vuln = name + ' ES VULNERABLE'
        else:
            vuln = name + ' NO ES VULNERABLE'
        return vuln
```

Este método recibe los parámetros necesarios para crear un *dataframe* y predecir si el usuario dado es vulnerable o no lo es. Explicaremos más adelante como funciona internamente el algoritmo y el código que hemos utilizado para instanciar el clasificador. Si recibimos un *1* el clasificador ha determinado que el usuario introducido es vulnerable. Si recibimos un *0*, se ha clasificado como no vulnerable.

En la página web, esta feature se observa como en la figura 5.

Prediccion de usuario critico

Isaac.Lozano ES VULNERABLE



The form consists of three input fields stacked vertically. The first field is a text box containing 'Isaac.Lozano'. The second field is a dropdown menu showing '150'. The third field is a dropdown menu showing '80'. Below these fields is a large, rounded rectangular button with a pink-to-purple gradient, containing the text 'ENVIAR' in white capital letters.

Figura 5: UI creada para determinar por IA si un usuario es vulnerable o no

5. Machine Learning

En esta práctica se nos facilitan dos ficheros *json*. Uno de ellos es un pequeño dataset de usuarios que están etiquetados en función de si son vulnerables o no. El otro fichero contiene otros usuarios con los mismos detalles pero no están etiquetados. Hemos usado el primero para entrenar los algoritmos de los que vamos a hablar a continuación para intentar etiquetar los segundos.

5.1. Decision Tree

El algoritmo *Decision Tree* es un algoritmo de aprendizaje supervisado que se usa para crear un modelo de árbol de decisiones a partir de un conjunto de datos de entrenamiento.

El árbol de decisiones se puede usar para predecir el valor de una variable objetivo a partir de un conjunto de variables de entrada. El código utilizado para generar el árbol y aplicarlo al conjunto de datos obtenido es el siguiente.

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(x_train, y_train)

y_predict = clf.predict(predecir)
c = 0
for i in y_predict:
    if i == 1:
        c += 1
print("Decision Tree classified as vulnerable", str(c), "users")
```

Se utiliza la librería *sklearn* para instanciar un clasificador que utiliza el algoritmo *Decision Tree*, le pasamos los datos de entrenamiento para que genere un árbol de decisiones con las condiciones necesarias para etiquetar el conjunto de datos dato. A continuación, le pasamos el conjunto de datos a predecir, y el método *predict* devuelve una lista con etiquetas correspondientes a lo que ha dicho el árbol de cada usuario proporcionado. El árbol generado se puede ver en la siguiente figura 6.

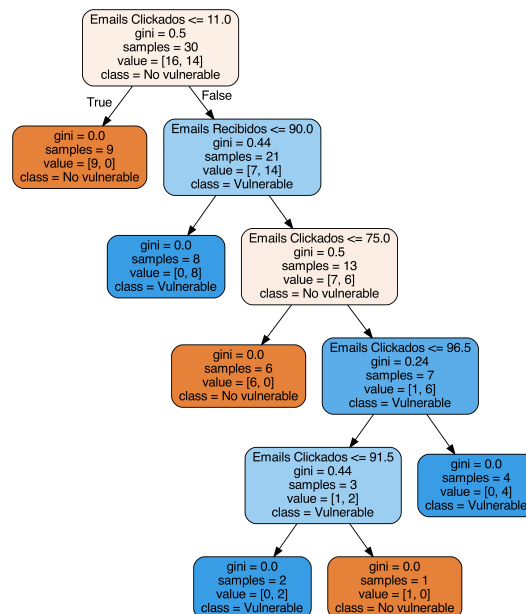


Figura 6: Árbol de decisiones creado por el algoritmo

5.2. Random Forest

El algoritmo de *Random Forest* es un algoritmo de aprendizaje automático que se puede usar tanto para la regresión como para la clasificación. Su funcionamiento se basa en la creación de un número específico de árboles de decisión, generalmente se usan entre 10 y 100. Cada uno de los árboles en el bosque vota por el valor que predice para un

determinado ejemplo, y el valor final que se asigna al ejemplo es la moda de los valores que votaron los árboles (es decir, el valor que más se repitió).

La principal diferencia entre el algoritmo de *Random Forest* y *Decision Tree* es que el primero crea un número específico de árboles y luego selecciona el que mejor se ajusta, mientras que el segundo solo crea un árbol.

El siguiente código se encarga de crear los árboles de decisión basados en los datos de entrada e intenta etiquetar de forma adecuada los datos de predicción.

```
rf_clf = RandomForestClassifier(max_depth=2, random_state=0, n_estimators=10)
rf_clf.fit(x_train, y_train)

# Predict data on test part
y_predict = rf_clf.predict(predecir)

c = 0
for i in y_predict:
    if i == 1:
        c += 1

print("Random forest classified as vulnerable", str(c), "users")
```

Primero se instancia el clasificador *RandomForestClassifier* con varios parámetros, como la profundidad máxima de los árboles generados. En la siguiente figura se puede observar uno de los árboles generados por el algoritmo.

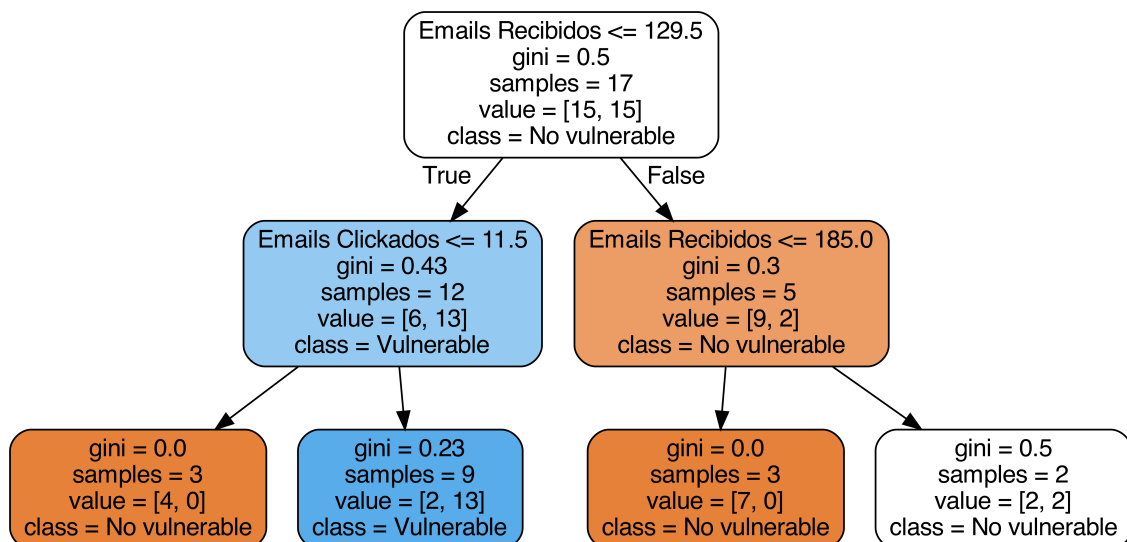


Figura 7: Árbol de decisiones creado por el algoritmo *Random Forest*

5.3. Regresión Linear

El algoritmo de regresión linear es un método para predecir valores futuros en función de datos anteriores. Se basa en la idea de que existe una relación lineal entre los datos y, por lo tanto, podemos ajustar una línea a los datos para hacer predicciones.

Para realizar este ejercicio, hemos tenido que dividir el conjunto de información etiquetado en dos partes. El código sigue el modelo de los dos ejercicios anteriores. Lo que hacemos es instanciar la clase *LinearRegression()*, ajustamos el modelo para los valores de entrenamiento e intentamos predecir los datos de valores no etiquetados.

```
data_x_train = data_x[:-20]
data_x_test = data_x[-20:]
data_y_train = data_y[:-20]
data_y_test = data_y[-20:]
regr = linear_model.LinearRegression()
regr.fit(data_x_train, data_y_train)

m = regr.coef_
b = regr.intercept_
x = data_x_test

data_y_pred = regr.predict(np.array(data_x_test))
for i in range(0, len(data_y_pred)):
    if data_y_pred[i] < 0.5:
        data_y_pred[i] = 0
    else:
        data_y_pred[i] = 1

print(mean_squared_error(data_y_test, data_y_pred))
print(accuracy_score(data_y_test, data_y_pred))
```

La gráfica resultante se puede ver en la siguiente figura 8.

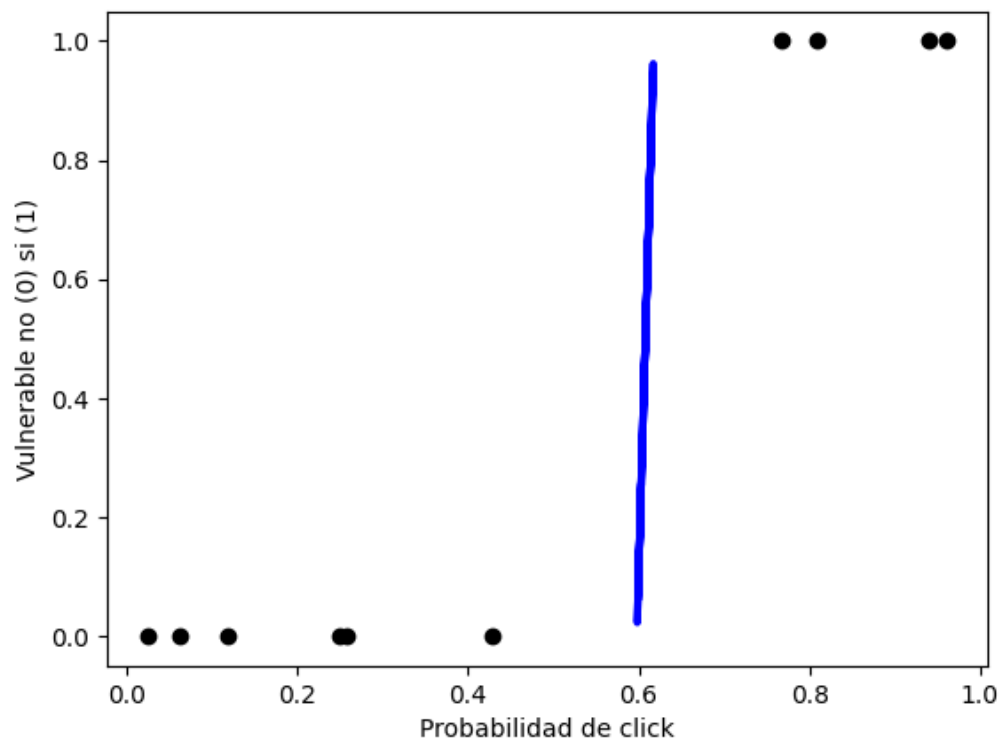


Figura 8: Gráfica generada por el algoritmo de regresión lineal

Como salida de los *print()* obtenemos lo siguiente:

Mean squared error: 0.35 Accuracy Regresión Lineal: 0.65