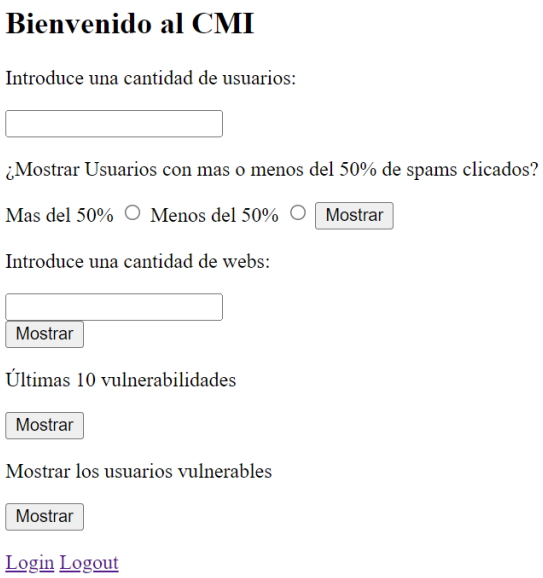
MEMORIA PRÁCTICA 2 SISTEMAS DE INFORMACIÓN

Marcos Bonilla Cubero m.bonillac.2018@alumnos.urjc.es 70068909F

Pablo Gracia Correa 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.



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:

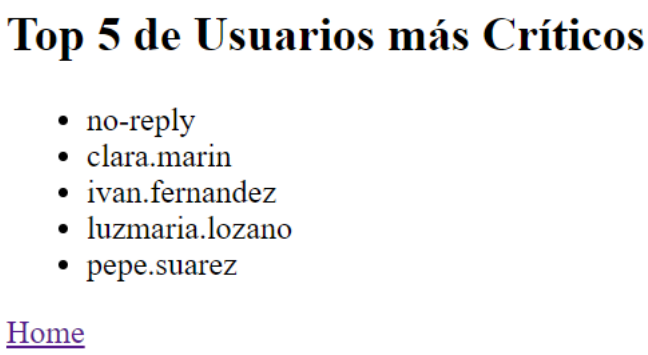
```python
@app.route('/topUsuariosCriticos/', methods=['POST'])
@login_required
def topUsuariosCriticos(usersMas=None, usersMenos=None):
    dfUsers = getFromDB('users')
    numero = request.form.get('numeroUsuarios')
    users = criticUsers(dfUsers, int(numero))[0]
    cincuentamas = request.form.get('50mas')
    cincuentamenos = request.form.get('50menos')
    if cincuentamas == 'on':
        usersMas = usersSpam('mas', dfUsers)
    if cincuentamenos == 'on':
        usersMenos = usersSpam('menos', dfUsers)
    return render_template('TopUsuariosCriticos.html', users=users, usersMas=usersMas, usersMenos=usersMenos)
```

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.

```python
def criticUsers(dfUsers, numero,vulnerables = ['5f4dcc3b5aa765d61d8327deb882cf99', '3bf1114a986ba87ed28fc
                    '276f8db0b86edaa7fc805516c852c889', '84d961568a65073a3bcf0eb216b2a576',
                    '0acf4539a14b3aa27deeb4cbdf6e989f', '1660fe5c81c4ce64a2611494c439e1ba',
                    'd16d377af76c99d27093abc22244b342', 'eb0a191797624dd3a48fa681d3061212',  '714ab9fbdad5
                    '4297f44b13955235245b2497399d7a93', '37b4e2d82900d5e94b8da524fbeb33c0','d0763edaa9d9bd
    usersVulnerable = []
    usersVulnerableCriticidad = []
    dfUsersCriticos = dfUsers.copy()
    dfUsersCriticos['criticidad'] = dfUsersCriticos['clicadosEmails'] / dfUsersCriticos['phisingEmails']
    dfUsersCriticos = dfUsersCriticos.sort_values("criticidad", ascending=False)
    for i in dfUsersCriticos.index:
        if dfUsersCriticos['contrasena'][i] in vulnerables and len(usersVulnerable) < numero:
            usersVulnerable.append(dfUsersCriticos['name'][i])
            usersVulnerableCriticidad.append(dfUsersCriticos['criticidad'][i])
    return usersVulnerable, usersVulnerableCriticidad
```

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:



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.

```python
@app.route('/topWebsVulnerables/', methods=['POST'])
@login_required
def topWebsVulnerables():
    dfLegal = getFromDB('legal')
    numero = request.form.get('numeroWebs')
    webs = outdatedWebs(dfLegal, int(numero))[0]
    return render_template('TopWebsVulnerables.html', webs=webs, numero=numero)
```

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:

```python
def outdatedWebs(dfLegal, numero):
    webDesactualizadas = []
    dfDesactualizados = dfLegal.copy()
    dfDesactualizados['desact'] = dfLegal.loc[:, ['cookies', 'aviso', 'proteccionDeDatos']].sum(axis=1)
    webs = []
    nPoliticas = []
    while len(webDesactualizadas) < numero:
        min = dfDesactualizados['desact'].min()
        dfAux = dfDesactualizados[dfDesactualizados['desact'] == min]
        dfAux = dfAux.sort_values('creacion')
        name = list(dfAux['name'].head(1))[0]
        webs.append(name)
        nPoliticas.append(min)
        webDesactualizadas.append(name)
        dfDesactualizados = dfDesactualizados.drop(dfDesactualizados[dfDesactualizados['name'] == name].index)
    return webs, nPoliticas
```

Y las mostramos en un template:

Ejercicio 3

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

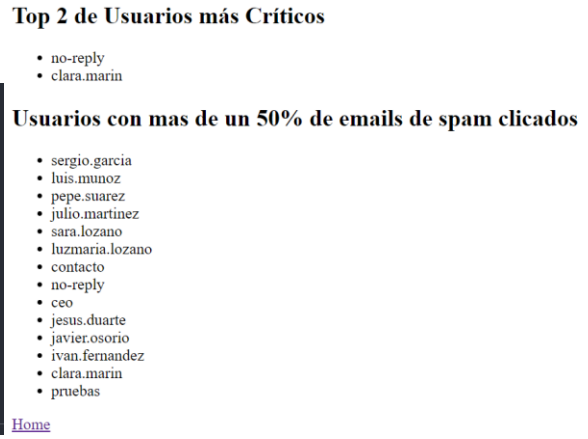
Como se puede ver en la página principal del CMI que mostramos al principio de la memoria hemos añadido unos botones para que se puedan elegir si mostrar los que tienen más del 50% o menos del 50%.

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):

```python
def usersSpam(duda, dfUsers):
    dfUsersCriticos = dfUsers.copy()
    dfUsersCriticos['criticidad'] = dfUsersCriticos['clicadosEmails'] / dfUsersCriticos['phisingEmails']
    if duda == 'mas':
        dfUsersCriticos = dfUsersCriticos[dfUsersCriticos['criticidad'] >= 0.5]
        return list(dfUsersCriticos['name'])
    elif duda == 'menos':
        dfUsersCriticos = dfUsersCriticos[dfUsersCriticos['criticidad'] < 0.5]
        return list(dfUsersCriticos['name'])
```

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:

```
{% if usersMas %}
<h2>Usuarios con mas de un 50% de emails de spam clicados</h2>
<ul>
    {% for user in usersMas %}
    <li>
        {{ user }}
    </li>
    {% endfor %}
</ul>
{% endif %} {% if usersMenos %}
<h2>Usuarios con menos de un 50% de emails de spam clicados</h2>
<ul>
    {% for user in usersMenos%}
    <li>
        {{ user }}
    </li>
    {% endfor %}
</ul>
```



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.

```python
@app.route('/lastVulnerabilities/', methods=['GET', 'POST'])
def lastVulnerabilities():
    response = rq.get('https://cve.circl.lu/api/last').json()
    vulnerabilities = []
    for resp in response:
        if len(vulnerabilities) < 10:
            vulnerabilities.append(resp['id'])
        else:
            break
    return render_template('lastVulnerabilities.html', vulnerabilities=vulnerabilities)
```

Ultimas 10 Vulnerabilidades descubiertas

- CVE-2022-20714
- CVE-2022-20716
- CVE-2022-28270
- CVE-2022-28272
- CVE-2022-28271
- CVE-2022-28273
- CVE-2022-28274
- CVE-2022-28276
- CVE-2022-28277
- CVE-2022-25786

Home

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:

```python
from flask_login import UserMixin

class User(UserMixin):

    def __init__(self, id, contrasena) -> None:
        self.id = id
        self.contrasena = contrasena
```

```python
from models.entities.user import User


class ModelUser():

    def login(self, user):
        allUsers = getFromDB('users')['name']
        if list(allUsers[user])[0]:
            return user
        else:
            return None

    @classmethod
    def get_by_id(self, id):
        allUsers = getFromDB('users')
        useri = allUsers[allUsers['name'] == id]
        user = User(list(useri['name'])[0], list(useri['contrasena'])[0])
        return user
```

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

Inicializamos el Login Manager:

```python
login_manager_app = LoginManager(app)
```

Y establecemos una super secret key:

```python
app.secret_key = 'super secret key'
```

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

```python
@login_manager_app.user_loader
def load_user(id):
    return ModelUser.get_by_id(id)
```

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

```python
@app.route('/login/', methods=['GET', 'POST'])
def login():
    if request.method == 'GET':
        return render_template('login.html')
    elif request.method == 'POST':
        name = request.form.get('name')
        password = request.form.get('password')

        dfUsers = (getFromDB('users'))
        password = hashlib.new('md5', password.encode())
        if list(dfUsers[dfUsers['name'] == name]['contrasena'])[0] == password.hexdigest():
            user = User(name, password.hexdigest())
            login_user(user)
            return redirect(url_for('index'))
        else:
            return render_template('login.html')
```

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:

```
@app.route('/logout/')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))
```

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:

```
@app.route('/usuariosVulnerablesIA/', methods=['GET'])
@login_required
```

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:

```
def error401(error):
    return render_template('notAutorized.html')


if __name__ == '__main__':
    app.secret_key = 'super secret key'
    app.register_error_handler(401, error401)
```

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:

```python
def get_users_prob():
    with open('Logs/users_IA_clases.json') as users:
        users = users.read()

    users = json.loads(users)
    users = pd.json_normalize(users['usuarios'])
    users_train_x = pd.DataFrame()
    users_train_x['porcentaje'] = users['emails_phishing_clicados'] / users['emails_phishing_recibidos']
    users_train_x['porcentaje'] = users_train_x['porcentaje'].fillna(0)
    users_train_y = users['vulnerable']

    with open('Logs/users_IA_predecir.json') as users:
        users = users.read()

    users = json.loads(users)
    users = pd.json_normalize(users['usuarios'])
    users_test_x = pd.DataFrame()
    users_test_x['porcentaje'] = users['emails_phishing_clicados'] / users['emails_phishing_recibidos']
    users_test_x['porcentaje'] = users_test_x['porcentaje'].fillna(0)
    users_name = users['usuario'].values
    print(users_name)

    return users_train_x, users_train_y, users_test_x, users_name
```

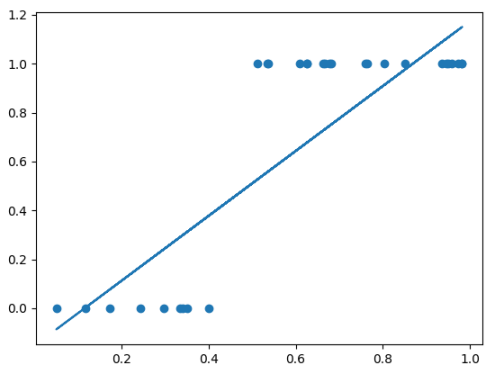
```python
def get_users_all():
    with open('Logs/users_IA_clases.json') as users:
        users = users.read()

    users = json.loads(users)
    users = pd.json_normalize(users['usuarios'])
    users_train_x = users[['emails_phishing_clicados', 'emails_phishing_recibidos']]
    users_train_y = users['vulnerable']

    with open('Logs/users_IA_predecir.json') as users:
        users = users.read()

    users = json.loads(users)
    users = pd.json_normalize(users['usuarios'])
    users_test_x = users[['emails_phishing_clicados', 'emails_phishing_recibidos']]

    users_name = users['usuario'].values
    print(users_name)

    return users_train_x, users_train_y, users_test_x, users_name
```

Regresión Lineal

```python
def linearRegresion():
    users_train_x, users_train_y, users_test_x, users_name = get_users_prob()
    regr = linear_model.LinearRegression()
    regr.fit(users_train_x, users_train_y)

    users_pred = regr.predict(users_test_x)
    users_res = users_pred.copy()
    for user in users_res:
        if user >= 0.5:
            users_res[np.argmax(users_res == user)] = 1
        else:
            users_res[np.argmax(users_res == user)] = 0

    print(users_res)
    plt.scatter(users_test_x, users_res)
    plt.plot(users_test_x.values, users_pred)
    plt.xticks()
    plt.yticks()
    plt.show()
    return users_res, users_name
```

En 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:

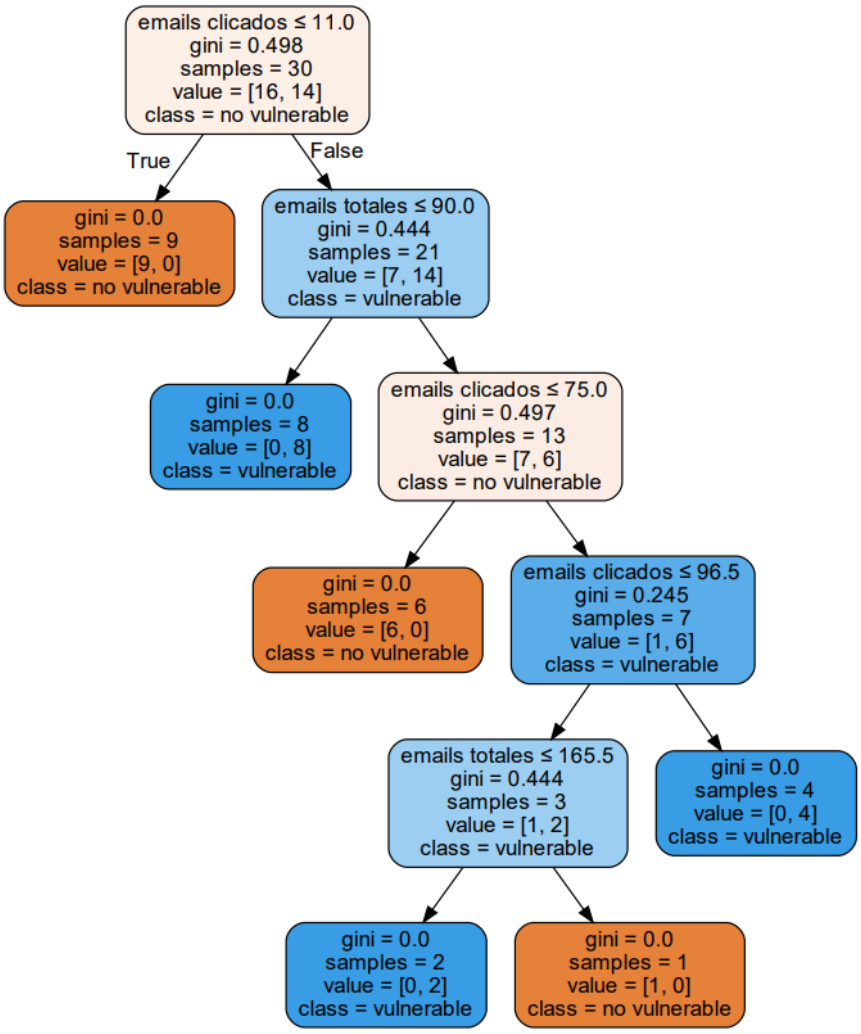


Decision Tree

El siguiente algoritmo que implementamos es el de Decision Tree:

```
def decisionTree():
    users_train_x, users_train_y, users_test_x, users_name = get_users_all()

    clf_model = tree.DecisionTreeClassifier()
    clf_model.fit(users_train_x, users_train_y)
    dot_data = tree.export_graphviz(clf_model, out_file=None,
                                    feature_names=['emails clicados', 'emails totales'],
                                    class_names=['no vulnerable', 'vulnerable'],
                                    filled=True, rounded=True,
                                    special_characters=True)
    graph = graphviz.Source(dot_data)
    graph.render('decisionTree.gv', view=True).replace('\\', '/')
    return clf_model.predict(users_test_x), users_name
```

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:



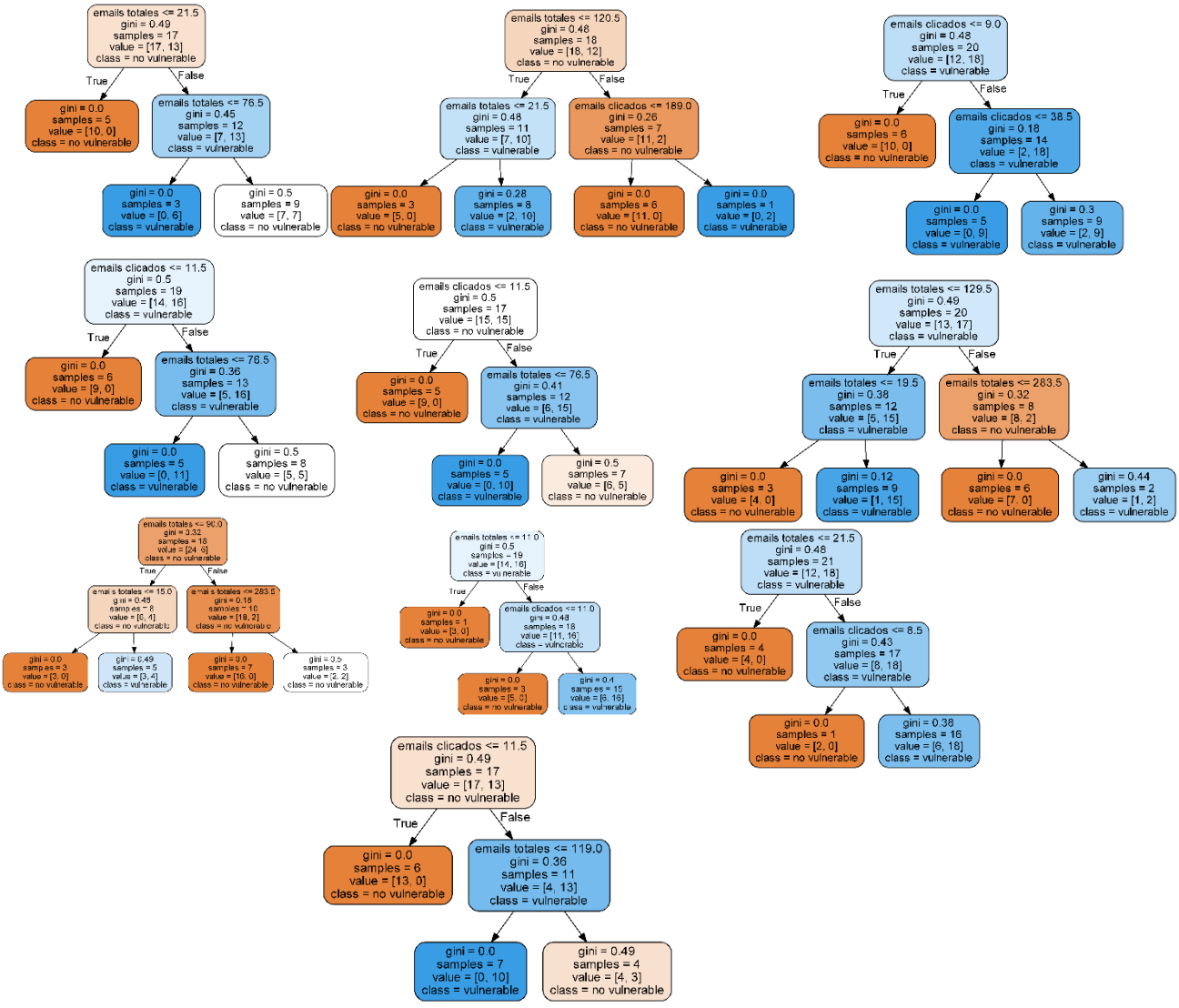
Random Forest

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.

```
def randomForest():
    users_train_x, users_train_y, users_test_x, users_name = get_users_all()

    clf = RandomForestClassifier(max_depth=2, random_state=0, n_estimators=10)
    clf.fit(users_train_x, users_train_y)

    for i in range(len(clf.estimators_)):
        estimator = clf.estimators_[i]
        tree.export_graphviz(estimator,
                             out_file='tree.dot',
                             feature_names=['emails clicados', 'emails totales'],
                             class_names=['no vulnerable', 'vulnerable'],
                             rounded=True, proportion=False,
                             precision=2, filled=True)
        call(['dot', '-Tpng', 'tree.dot', '-o', 'tree' + str(i) + '.png', '-Gdpi=600'])

    return clf.predict(users_test_x), users_name
```

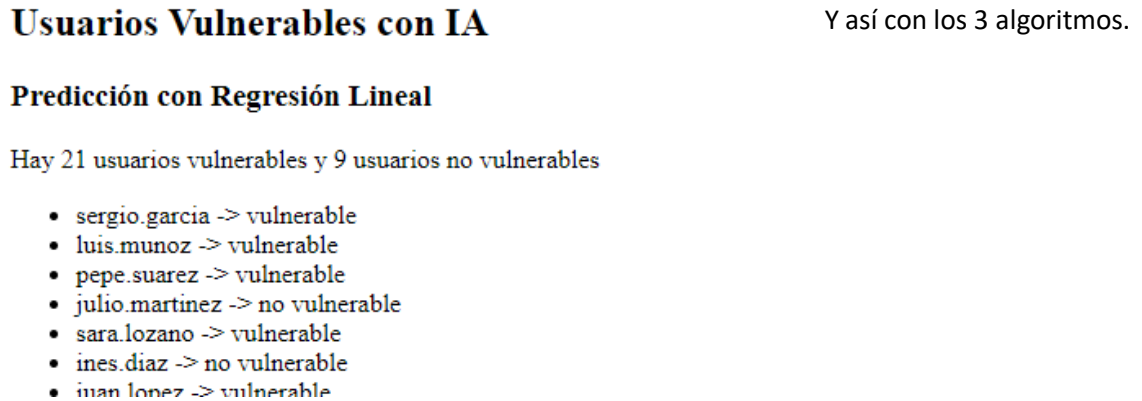
Al igual que en los otros algoritmos usamos una librería de sklearn, primero entrenamos y luego predecimos.

Se 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:

```python
@app.route('/usuariosVulnerablesIA/', methods=['GET'])
@login_required
def usuariosVulnerablesIA():
    regresion, names = linearRegresion()
    decTree = decisionTree()[0]
    forest = randomForest()[0]
    print(regresion, decTree, forest)

    return render_template('usuariosVulnerablesIA.html', names=names, regresion=regresion,
                           regresionCount=np.sum(regresion), decTree=decTree, decTreeCount=np.sum(decTree),
                           forest=forest, forestCount=np.sum(forest))
```



Usuarios Vulnerables con IA

Predicción con Regresión Lineal

Hay 21 usuarios vulnerables y 9 usuarios no vulnerables

- sergio.garcia -> vulnerable
- luis.munoz -> vulnerable
- pepe.suarez -> vulnerable
- julio.martinez -> no vulnerable
- sara.lozano -> vulnerable
- ines.diaz -> no vulnerable
- juan.lopez -> vulnerable

Y así con los 3 algoritmos.

GitHub

https://github.com/pgraciac/PracticaSI