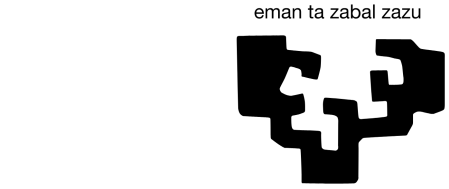


Web Scraping

Minería de Datos

Josu Barrutia

14 de enero de 2023

Índice

1. Data Scraping **1**

2. Funcionamiento **1**
2.1. Extrayendo lugares para escalar 1

3. Crawlers **3**
3.1. Extrayendo todas las vías de escalada 4

4. API **6**

5. Aplicaciones **6**
5.1. Generando nuestro modelo 7

1. Data Scraping

El valor de la posesión de datos ha generado una gran demanda en la extracción de información de cualquier tipo de fuente. Como la cantidad de información a la que se tiene acceso es inmensa, surgen una nueva práctica para la extracción automática de datos, el *Data Scraping*. *Scraping*, del inglés, hace referencia al rascado/extracción de información de manera automatizada. El objetivo del *Data Scraping* es la extracción de información de fuentes de datos no estructuradas para darles un formato adecuado para su propósito (CSV, JSON, XML). Existen distintas fuentes como puede ser aplicaciones, vídeos, audio o sitios web, nos centraremos es este último que recibe un nombre propio *Web Scraping*.

2. Funcionamiento

El *Web Scraping* es un tipo de *Data Scraping* que se centra en la extracción de información de un sitio web. En primer lugar, se debe indicar el URL del sitio web del que se va a extraer la información, una dirección URL apunta a un único recurso.

Es necesario entender el protocolo HTTP (HyperText Transfer Protocol), pues es el encargado de transmitir datos en la World Wide Web. Cuando se escribe una dirección URL en el navegador, se está enviando una solicitud HTTP a un servidor web, este suele contestar con un contenido HTML que es renderizado por el navegador. Así, el usuario solo ve el contenido HMTL renderizado, sin embargo, para scrapear la web es necesario obtener el código fuente o HTML. Existen varios tipos de solicitudes HTTP, para realizar el scraping se utilizará GET, el método HTTP GET solicita una representación del recurso especificado. Las solicitudes que usan GET solo deben usarse para recuperar datos (no deben incluir datos) [1].

A continuación, se realiza un filtrado por medio de localizadores para extraer la información deseada del código fuente. Los principales localizadore, y, por ello, es necesario tener una nociones básicas, son las etiquetas en HTML, selectores CSS o XPath, un lenguaje que te permite localizar elementos específicos de una página, se puede utilizar para navegar a través de elementos y atributos en un documento XML. Para XPath, un documento XML es como un árbol, que está compuesto por dos conceptos. Por un lado, la representación de distintos tipos de nodos; por el otro, las posibles relaciones que existen entre estos nodos [4].

Por último, se almacena la información en un formato estructurado. Uno de los principales formatos para el almacenamiento es el CSV, se trata de un formato de archivo que se utiliza para almacenar datos tabulares en forma de texto. Es un formato muy sencillo en el que cada fila de la tabla se representa en una línea del archivo y las columnas se separan por comas. Otro formato muy usado es el JSON (JavaScript Object Notation), es un formato de texto pensado para el intercambio de datos. Su sintaxis está basada originalmente en la sintaxis de JavaScript, pero realmente es independiente de cualquier lenguaje de programación [2].

2.1. Extrayendo lugares para escalar

Vamos a extraer todas las localizaciones de escalada en Euskadi que contempla la página web theCrag. Para ello, vamos a crear un script en Python que scrapee la página y almacene las localizaciones en un fichero txt.

Utilizaremos las librerías BeautifulSoup que se utiliza para extraer datos de documentos HTML y XML (lenguaje de marcado extensible, cuya finalidad es definir una sintaxis para la codificación de documentos, que tanto los usuarios como las propias máquinas en sí puedan ser capaces de leer. De hecho, la extensión .docx incorpora la x del final por el uso de XML [3]). Proporciona una interfaz simple y rápida para buscar y navegar por el árbol de etiquetas del documento.

Y la librería requests que se utiliza para realizar solicitudes HTTP a un servidor y recibir su respuesta. Nosotros la utilizaremos para realizar una solicitud get a la página en cuestión.

```
1 from bs4 import BeautifulSoup
2 import requests
```

Código 1: Importación de librerías

Realizamos una solicitud get al url en cuestión que devuelve un objeto requests.Response que posee varias propiedades y métodos interesantes, nosotros nos quedaremos con content, que se traduce en el código fuente de la página. Pasaremos este código fuente al constructor de BeautifulSoup, para crear un objeto BeautifulSoup con el que fácilmente extraeremos la información que queremos por medio de expresiones regulares.

```
url = 'https://www.thecrag.com/es/escalar/spain/pamplona-vitoria-gasteiz-area' #Realmente se
    ↪ refiere al país vasco
page = requests.get(url)
html = BeautifulSoup(page.content, 'html.parser')
```

Código 2: Conseguir código fuente

Empezamos a filtrar por la etiqueta div de class=’node-listview hide-archived’ y buscamos todas las entradas span de class=’primary-node-name’. Con esto conseguiremos una lista de objetos de etiquetas span de clase “primary node name” que se encuentren dentro del div “node-listview hide-archived”, que sigue siendo no más que código html.

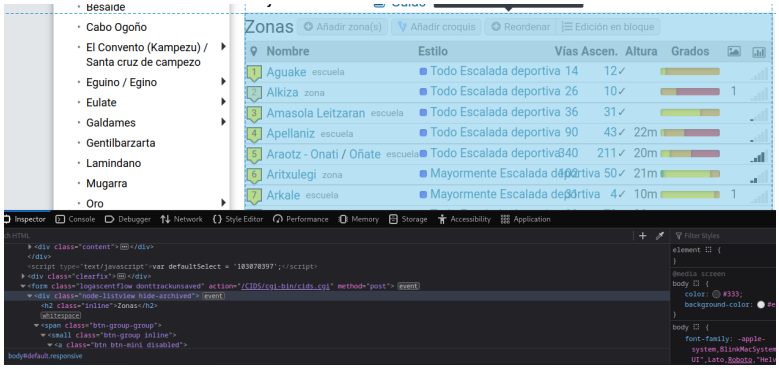


Figura 1: Proceso para obtener localizadores

```
entradas = html.find('div', {'class': 'node-listview hide-archived'})
entradas = entradas.find_all('span', class_='primary-node-name')
```

Código 3: Scrapeado

Lo que nos interesa a nosotros es la información “human-readable” que contienen dichas etiquetas. Para ello, iteraremos por los atributos text y los almacenaremos en una lista que imprimiremos por la salida estándar y almacenaremos en un fichero txt.

```
vias = list()

for i in entradas:
    vias.append(i.text)
print(vias)

with open("localizaciones.txt", "w") as output:
    output.write(str(vias))
```

Código 4: Almacenamiento

```
['Aguake', 'Alkiza', 'Amasola Leitzaran', 'Apellaniz', 'Araotz - Onati', 'Aritxulegi', 'Arkale
    ↪ ', 'Atauri', 'Atxarte', 'Baltzola', 'Besaide', 'Cabo Ogoño', 'El Convento (Kampezu)', '
    ↪ Eguino', 'Eulate', 'Galdames', 'Gentilbarzarta', 'Lamindano', 'Mugarra', 'Oro', '
    ↪ Pagasarri', 'Platako Farola', 'Pikoketa', 'Los Pinones', 'Piugaz', 'Jaizkibel', 'Puntas'
    ↪ , 'San Marcos', 'Santa Barbara', 'Sobrón', 'Subijana', 'Untzillatx-Sur', 'Urduliz', '
    ↪ Urgull', 'Villanueva de Valdegovía', 'Zazpiturri', 'Txindoki', 'Otoio', 'Trucios /
    ↪ Turtzioz']
```

Código 5: Output

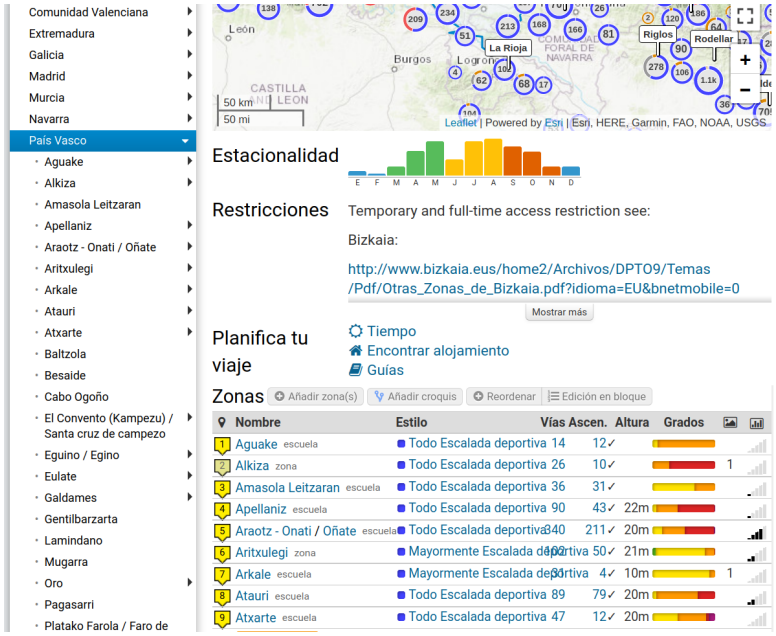


Figura 2: Página web del scrapeo

Evidentemente, existen aplicaciones que realizan todo este trabajo por ti, pero lo interesante en este proyecto es realizarlo a más bajo nivel, eso sí valiéndonos de algunas librerías que facilitan algo el trabajo.

3. Crawlers

Un crawler es un programa que recorre la web de forma automática para recopilar información. Estos ´´crawlers" se denominan también arañas, arañas web, robots o bots. Los crawlers se utilizan comúnmente para recolectar páginas web para luego indexarlas en motores de búsqueda, para crear copias de seguridad de sitios web, para recopilar estadísticas sobre el tráfico del sitio web, o para recolectar datos para análisis de mercado. Son el proprio Google, DuckDuckGo y demás indexadores los que utilizan estos bots para poder ofrecer pagínas web a las que acceder.

En general, un crawler comienza con una cola de URLs para visitar y recupera cada página en la cola. A medida que visita cada página, extrae enlaces a otras páginas y agrega estos enlaces a la cola de URLs. El proceso continúa hasta que se agoten las URLs en la cola. Los crawlers pueden ser configurados para seguir sólo ciertos tipos de enlaces, evitando así la recuperación de páginas que no son relevantes para el objetivo del rastreador.

Esta búsqueda es comparable a una búsqueda en anchura que se realiza en estructuras de datos como grafos para encontrar posibles caminos y elegir el más corto. Realmente la web es un grafo, con páginas web (identificadas por URLs) como vértices y referencias de una web a otra como aristas, pues están conectando dos webs la actual y la referenciada.

```
# Google Search Engine Robot
# ==========================
User-agent: Googlebot
Allow: /?_escaped_fragment_

Allow: /*?lang=
Allow: /hashtag/*?src=
Allow: /search?q=%23
Allow: /i/api/
Disallow: /search/realtime
Disallow: /search/users
Disallow: /search/*/grid

Allow: /*?ref_src=
Allow: /*?src=
Disallow: /*?
Disallow: /*/followers
Disallow: /*/following

```

```
Disallow: /account/deactivated
Disallow: /settings/deactivated
```

Código 6: Robots.txt de Twitter para el crawler de Google

El archivo robots.txt es un archivo que se suele colocar en la raíz de un sitio web y que contiene instrucciones para los crawlers (generalmente los pertenecientes a motores de búsqueda como Google) sobre qué páginas o secciones del sitio web deben o no ser rastreadas. El formato estándar para el archivo robots.txt es muy sencillo y consta de varias líneas de texto que indican qué usuarios-agentes (es decir, los nombres de los crawlers específicos) están permitidos o denegados en qué partes del sitio.

Esta práctica es bastante utilizada para que los motores de búsqueda no indexen secciones que son privadas y que el creador de la página no quiere que sean públicos. No obstante, aunque un crawler respete las instrucciones del archivo robots.txt, esto no significa necesariamente que la información de esa sección no será rastreada o indexada, pues se trata de un protocolo meramente consultivo, luego no previene el scrapeo de su contenido.

Los Scrapers y Crawlers suelen trabajar en dupla, siendo el crawler el que guía al scraper por las distintas páginas web mientras que el scraper va extrayendo la información pertinente de cada una de ellas.

3.1. Extrayendo todas las vías de escalada

Vamos a extraer todas las vías de todos los sectores de todas las localizaciones del País Vasco que contempla la página theCrag. Para ello, no nos vale con scrapear la página en cuestión. Como la página está dividida en secciones, debemos obtener rutas potenciales donde encontraremos información sobre cada vía. Así, implementaremos un crawler que obtenga todas las rutas que redireccionan a áreas que contengan información sobre las vías del País Vasco.

Empezamos desde la página theCrag para la que hacemos un scrapeo e insertaremos en una cola los URLs que nos interesen. Para cada enlace en la cola, realizamos el mismo proceso hasta que no queden enlaces en la cola.

El scrapeo se basa en filtrar por el cuadro que contiene la información relevante, para cada div de clase 'route', buscamos el nombre de la vía y su respectiva graduación, esto se conseguirá filtrando por el span de clase 'primary-node-name' para conseguir el nombre de las vías y el span de clase 'gb3' para conseguir su respectiva graduación. Para filtrar por el cuadro que contiene toda la información relevante, y para cada div de clase 'name' buscaremos todas las etiquetas 'a' que contienen los enlaces. No obstante, estos enlaces son lo que llamamos relativos, por lo que tendremos que concatenarlo con el enlace previo, el de la iteración anterior.

Finalmente, crearemos un pandas dataFrame y almacenaremos la información en un csv.

```python
from bs4 import BeautifulSoup
import requests
from urllib.parse import urljoin
import pandas as pd


vias = list()
graduaciones = list()

def scrap(url):
    page = requests.get(url)
    html = BeautifulSoup(page.content, 'html.parser')

    entrada = html.find('div', {'class': 'node-listview hide-archived'})
    if entrada is  None:
        return

    info_via = entrada.find_all('div', {'class': 'route'})
    if info_via is None:
        return

    for i in info_via:
        nombre_via = i.find('span', {'class': 'primary-node-name'})
        graduacion = i.find('span', {'class': 'gb3'})
        if(nombre_via is None):
            continue
        nombre_via=nombre_via.text
        if(graduacion is None):
            continue
```

```
        graduacion=graduacion.text
        print(nombre_via,' : ' , graduacion)
        vias.append(nombre_via)
        graduaciones.append(graduacion)


class Crawler:

    def __init__(self, urls=[]):
        self.visited_urls = []
        self.urls_to_visit = urls

    def download_url(self, url):
        return requests.get(url).text

    def get_linked_urls(self, url, html):
        soup = BeautifulSoup(html, 'html.parser')
        cont = soup.find('div', {'class': 'node-listview hide-archived'})
        if cont is not None:
            for i in cont.find_all('div', {'class': 'name'}):
                link = i.find('a')
                if link is not None:
                    path = link.get('href')
                    if path is None:
                        continue
                    elif path.startswith('/'):
                        path = urljoin(url, path)
                    yield path

    def add_url_to_visit(self, url):
        if url not in self.visited_urls and url not in self.urls_to_visit:
            self.urls_to_visit.append(url)

    def crawl(self, url):
        html = self.download_url(url)
        for url in self.get_linked_urls(url, html):
            self.add_url_to_visit(url)

    def run(self):
        while self.urls_to_visit:
             url = self.urls_to_visit.pop(0)
             print(url)
             scrap(url)
             self.crawl(url)
             self.visited_urls.append(url)

if __name__ == '__main__':
    Crawler(urls=['https://www.thecrag.com/es/escalar/spain/pamplona-vitoria-gasteiz-area']).
    ↪ run()
    df = pd.DataFrame({'vias': vias, 'graduaciones': graduaciones})
    df.to_csv('vias.csv', index=False)
```

Código 7: Crawling and Scraping

Este programa se demora una gran cantidad de tiempo por la inmensa cantidad de solicitudes que realiza, que, aunque no sean solicitudes de muchos bytes, el factor limitante es la velocidad de transmisión.

Al ejecutar el programa veremos la página que visita y las vías que extrae de ella, el output es larguísimo, por lo que solo podré mostrar un trozo muy pequeño.

```
https://www.thecrag.com/es/escalar/spain/urduliz/area/5395862526
Ezpata  :  6a+
Begihandi  :  6c
https://www.thecrag.com/es/escalar/spain/urduliz/area/5381018424
Significado de grupo  :  6b
Win Hof  :  6a+
Clavada  :  6a+
Placa placa  :  6b+
https://www.thecrag.com/es/escalar/spain/urduliz/area/5393806953
Aguya  :  6b
Kienpica  :  6c
Gure Lobak  :  6b
```

Código 8: Ejemplo del output

Analizando todo el output que genera el programa podemos ver como el recorrido que sigue por las páginas web es el mismo que el que realiza el algoritmo bread-first-search en un grafo. Accede a todas las páginas web que están a distancia uno del origen y, cuando a visitado todas las de distancia uno, sigue con las de distancia dos.

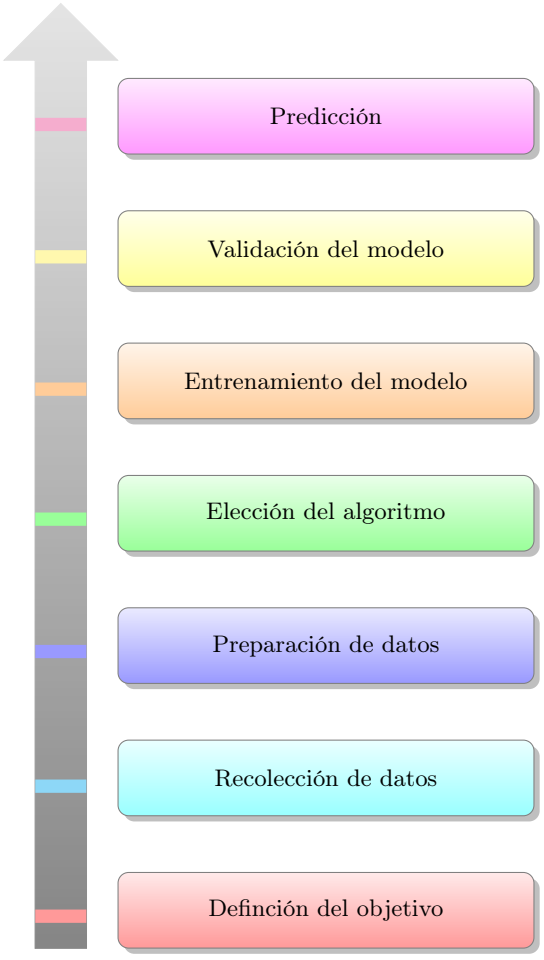
4. API

API (Application Programming Interface) es un conjunto de protocolos, reglas y herramientas para la construcción de aplicaciones. Para entenderlo, podemos compararlo con las interfaces que se aprende a utilizar en Java, que ofrecen los métodos que son posibles realizar para una determinada clase. Una API nos ofrece un contrato de servicio entre dos aplicaciones, definiendo la manera en que se comunican mediante solicitudes (queries) y respuestas. La respuesta de las APIs suelen seguir un formato JSON o XML, por lo que crear una base de datos con la información de una API resulta más sencillo que cuando se scrapea. Sin embargo, el verdadero reto con las APIs es conectarte a ellas. En muchos casos es necesaria una API Key que se utiliza para suprimir el acceso no autorizado a una API. Esto se logra mediante la verificación de la identidad del usuario, el control de los límites de uso, la detección de uso anormal y la autenticación de solicitudes.

5. Aplicaciones

La obtención de datos mediante una API o realizando web Scraping tiene infinidad de aplicaciones como el estudio del mercado, comparación de precios, ... Nosotros nos centraremos en su esencialidad para la obtención de modelos de machine learning.

El primer paso del machine learning es entender el problema de clasificación. Por ejemplo, si se quiere clasificar las complicaciones que pueden darse tras un infarto de miocardio, es necesario entender mínimamente diversas métricas que está relacionadas con ellas, presión arterial, electrocardiograma, ... para hacer un buen uso de ellas, pues serán nuestras variables predictoras.



El segundo paso es recolectar los datos, es la etapa en la que se ha hecho hincapié, extrayendo datos de páginas con API y sin ellas. Es una fase vital que condicionará fases posteriores como el entrenamiento y todas las posteriores. Podemos verlo como el "combustible" del aprendizaje automático, se usarán para el entrenamieno y validación del modelo. Si los datos no son adecuados ni de buena calidad (es importante que el dataset comprenda instancias de todas las clases posibles y que haya un buen equilibrio, que el 95 % de clases sea lluvia y el resto no lluvia penalizaría bastante la tasa de acierto del modelo).

La preparación de los datos puede resultar una de las fases más tediosas, pues se trata de depurar los datos, eliminando instancias cuyas variables estén incompletas, discretización de la clase si queremos enfocar el problema en uno de clasificación y no de regresión, darles el formato adecuado, combinarlos con otros de otras fuentes, normalizarlos ... Generalmente el web scraping generá más problemas de este tipo, siendo las APIs mucho más cómodas.

La selección del algoritmo es suele ser una fase determinante, pues la selección incorrecta puede conducir a resultados pobres o inesperados. Por lo tanto, es clave seleccionar el algoritmo adecuado para el problema en cuestión para asegurar que se obtengan los resultados deseados. La elección del algoritmo correcto también puede ahorrar tiempo y recursos al acortar el tiempo de entrenamiento y mejorar la precisión de los resultados.

Para el entrenaiemto del modelo se suelen hacer particiones del dataset, de manera que un porcentaje del total de instancias se utilizan para el entrenamiento del modelo y las demás se utilizan para la validación del modelo generado tras el entrenamiento.

Tras generar un modelo, se valida para observar su rendimiento ante las instancias que no han participado en el entrenamiento. Existen varias técnicas de validación para las que la forma de reservar instancias cambia. Cabe destacar que un modelo puede funcionar muy bien sobre las instancias con que ha sido entrenado, pero no tan bien sobre instancias nuevas que no han participado en el entrenamiento (overfitting), es por esto por lo que se emplean técnicas de validación como Holdout, K-fold-cross-validation, ... pero nunca usamos todo el dataset como entrenamiento. Muchas veces se reserva una tercera partición, para realizar tunning sobre parámetros de nuestro algoritmo y observar con cuál rinde mejor. De hecho, varias librerías que implementan algoritmos de clasificación no supervisada incluyen esta función.

Por último, cuando la tasa de acierto y demás métricas hayan cumplido nuestras expectativas, podremos proceder a predecir ejemplos.

5.1. Generando nuestro modelo

Para la escalada en roca es vital conocer la probabilidad de precipitación del sitio al que vas a ir, pues escalar sobre una vía humeda es inviable. Para conocer dicha probabilidad, vamos a generar un modelo basado en el clasificador Naive Bayes para predecir si va a llover o no, dadas unas métricas del tiempo como la temperatura, presión atmosférica, ... que serán nuestras variables predictoras.

Para ello, vamos a extraer datos los datos del tiempo de todos los días del año 2022 en Euskadi, mediante la API de worldWeatherOnline. Tenemos que ir haciendo queries de mes a mes, pues tiene un máximo de longitud en su respuesta. Nos devuelve la información del tiempo cada 3 horas para cada día. La respuesta contiene varios parámetros como la velocidad del viento, temperatura, presión atmosférica, ...

La respuesta es un archivo JSON, del que extraeremos únicamente la información que nos es conveniente, por lo que al terminar no tendremos que descartar variables predictoras y apenas tendremos que limpiar el dataset. Creamos un pandas dataFrame y lo almacenamos en un csv en nuestro disco local por si acaso.

La única limpieza del dataset que tenemos que hacer es un casting de los datos, para, a continuación, poder discretizar la variables "precipMM", que previamente medía la cantidad de precipitación por milimetro cuadrado, no obstante, ahora nos dirá si llueve o no.

Normalizamos el dataset y realizamos un holdout 80-20 para validar el modelo que vamos a generar, para ello dividimos las instancias en entrenamiento y test, separando también la variable clase, "precipMM".

Por último, creamos el modelo, que será un Naive Bayes y lo entrenamos. Al predecir las instancias de test obtenemos una tasa de acierto de alrededor 0.89, lo cual está bastante bien.

Había realizado la prueba con otras APIs que solo ofrecían los datos del último mes, por lo que el dataset quedaba muy pobre, sin instancias del tiempo en verano (dataset desequilibrado), así que la predicción no sería fiel a la realidad. Aun y todo, me sorprende las pocas instancias de no lluvia que hay en las instancias de test.

```
import requests
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
from mlxtend.plotting import plot_decision_regions

tempC = []
windspeedKmph = []
precipMM = []
humidity = []
visibility = []
pressure = []
cloudcover = []
HeatIndexC = []
DewPointC = []
WindChillC = []
WindGustKmph = []
FeelsLikeC = []
uvIndex = []

def main():
    for i in range(1, 12):
```

```
        getData('2022-%s-01' % i, '2022-%s-01' % (i+1)) #Llamamos a la funcion getData para obtener los datos de la api para cada mes
    df = createDataFrame()
    print(df.head())
    df.to_csv('weather2022.csv') # guardamos el pandas Dataframe en un csv por si acaso
    #La unica depuracion que tenemos que hacer en el dataframe
    df['precipMM'] = df['precipMM'].astype(float) #Convertimos la columna precipMM a float
    df = df.astype('int64') #Convertimos todas las columnas a int64
    # Discretizamos la clase
    clase = df['precipMM'] > 0
    df.loc[clase, 'precipMM'] = 1
    df.loc[~clase, 'precipMM'] = 0
    #Como se trata de un dataset extraido de una api y creado por nosotros ya sabemos que no tenemos que hacerle un tratamiento previo
    #Normalizamos los datos
    df_normalizado = df.loc[:, df.columns != 'precipMM'].apply(normalize, axis=0)
    df_normalizado['precipMM'] = df['precipMM']
    #Hacemos un Hold-out 80-20 para entrenamiento y test
    X_train, X_test, y_train, y_test = train_test_split(df_normalizado.drop(['precipMM'], axis = 1), df_normalizado['precipMM'], test_size=0.2, random_state=0)
    #Entrenamos el modelo
    model = GaussianNB()
    model.fit(X_train, y_train)
    #Testeamos
    pred = model.predict(X_test)
    print("Tasa de acierto:",metrics.accuracy_score(y_test, pred))
    #Una graficas
    ConfusionMatrixDisplay.from_estimator(model, X_test, y_test).plot()


def getData(data1, data2):
    url = 'http://api.worldweatheronline.com/premium/v1/past-weather.ashx?key=66ba03ff2d45466c8aa102127231201&q=43.31283, -1.97499&format=json&date=%s&enddate=%s&tp=3' % (data1, data2)
    response = requests.get(url)
    json = response.json()
    data = json['data']['weather']

    for i in range(len(data)):
        for j in range(0,8):
            tempC.append(data[i]['hourly'][j]['tempC'])
            windspeedKmph.append(data[i]['hourly'][j]['windspeedKmph'])
            precipMM.append(data[i]['hourly'][j]['precipMM'])
            humidity.append(data[i]['hourly'][j]['humidity'])
            visibility.append(data[i]['hourly'][j]['visibility'])
            pressure.append(data[i]['hourly'][j]['pressure'])
            cloudcover.append(data[i]['hourly'][j]['cloudcover'])
            HeatIndexC.append(data[i]['hourly'][j]['HeatIndexC'])
            DewPointC.append(data[i]['hourly'][j]['DewPointC'])
            WindChillC.append(data[i]['hourly'][j]['WindChillC'])
            WindGustKmph.append(data[i]['hourly'][j]['WindGustKmph'])
            FeelsLikeC.append(data[i]['hourly'][j]['FeelsLikeC'])
            uvIndex.append(data[i]['hourly'][j]['uvIndex'])

def createDataFrame():
    df = pd.DataFrame({'tempC': tempC,
                       'windspeedKmph': windspeedKmph,
                       'precipMM': precipMM,
                       'humidity': humidity,
                       'visibility': visibility,
                       'pressure': pressure,
                       'cloudcover': cloudcover,
                       'HeatIndexC': HeatIndexC,
                       'DewPointC': DewPointC,
                       'WindChillC': WindChillC,
                       'WindGustKmph': WindGustKmph,
                       'FeelsLikeC': FeelsLikeC,
                       'uvIndex': uvIndex})
    return df

def normalize(x):
    return((x-min(x)) / (max(x) - min(x)))

```

```
95 if __name__ == '__main__':
96     np.random.seed(2)
97     main()
```

Código 9: Naive Bayes para predecir la precipitación

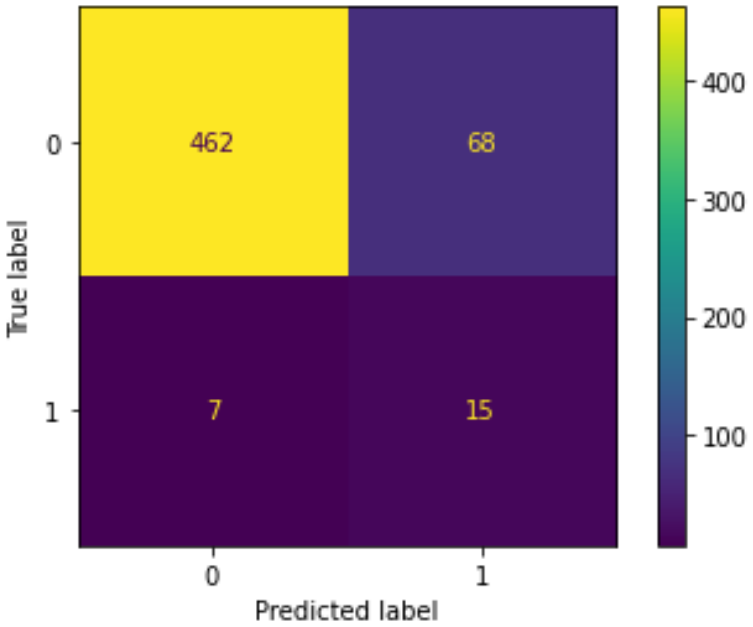


Figura 3: Matriz de confusión

Referencias

[1] Get - http: Mdn. https://developer.mozilla.org/es/docs/Web/HTTP/Methods/GET. Accessed: 2023-1-2.

[2] Json. https://www.mclibre.org/consultar/informatica/lecciones/formato-json.html. Accessed: 2023-1-2.

[3] Muyinteresante.es. https://www.muyinteresante.es/tecnologia/articulo/que-es-y-como-abrir-un-archivo-xml-201608322902. Accessed: 2023-1-2.

[4] Octoparse. https://www.muyinteresante.es/tecnologia/articulo/que-es-y-como-abrir-un-archivo-xml-20160 Accessed: 2023-1-2.