PRÁCTICAS (APARTADO III)

Análisis, Diseño y Procesamiento de Datos Aplicados a las Ciencias y a las Tecnologías(ADP)

Máster Universitario en Inteligencia Computacional e Internet de las Cosas

Universidad de Córdoba, EPSC

2022/2023

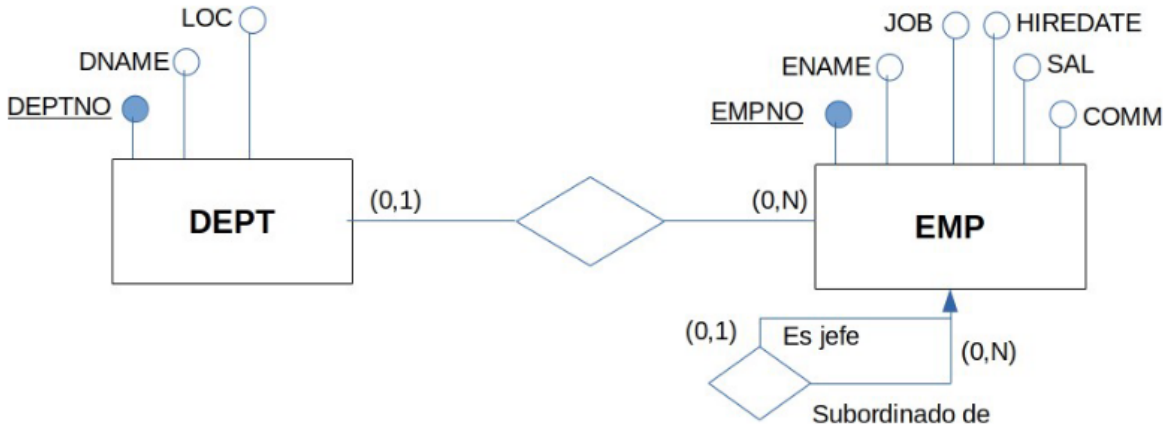
Autor:

Antonio Gómez Giménez (i72gogia@uco.es)

Índice:

FASE 1: SOLUCIÓN NOSQL MONGODB

El objetivo de esta fase es afianzar los conocimientos impartidos en la parte teórica de la asignatura sobre el uso de la solución NoSQL MongoDB. Para ello se pedía realizar el siguiente esquema entidad-interrelación:



Para ello se realizaron los dos siguientes pasos, la creación del esquema y carga de la información, y posteriormente, la realización de diferentes consultas para comprobar el correcto funcionamiento de la misma.

1. Apartado 1 y 2 - Creación del esquema/Carga de información:

Para la realización y carga de datos se planificó usar como estructura las siguientes dos colecciones:

La colección departamento donde nos encontraríamos el siguiente documento de ejemplo:

DEPT:
_id: nº ID
DNAME: "Nombre del Departamento"
LOC: "Localización del Departamento"

EMP:
_id: nº ID
ENAME: "Nombre del Empleado"
JOB: "Trabajo del empleado"
HIREDATE: Isodate con la fecha de contratación
SAL: Número con el salario del empleado
COMM: "atributo cuyo valor va a ser siempre temp"
DEPARTAMENTO: id del departamento que pertenece el empleado
Es_JEFE: array con el id de los empleados de los cuales es jefe
Es_subordinado: id del empleado que es jefe para este empleado

Una vez planteada la estructura general, se realiza el código para la conexión de la bd y creación de ambas colecciones.

```python
parser = argparse.ArgumentParser(description='Programa para realizar la practica 1 y sus 3 modos',)
parser.add_argument("-m", "--mode", help="Modo de funcionamiento para cada apartado", action="store", dest="modo", type=int)
parametros = parser.parse_args()

# Aquí procesamos lo que se tiene que hacer con cada argumento
if parametros.modo == 0:
    print("Creación del esquema")

    # conexión con la base de datos en la nube
    client = pymongo.MongoClient("mongodb+srv://AntonioGG:patata@cluster0.uoqzgcw.mongodb.net/?retryWrites=true&w=majority")
    #client = pymongo.MongoClient('mongodb://localhost:27017')

    # vemos las bases de datos existentes
    database_list = client.list_database_names()
    print("Nombres de bd:", database_list)

    #damos la opción de borra alguna bd
    name = input("Introduce el nombre de la base de datos a borrar: ")
    if name == "":
        print("No se elimina ninguna bd")
    else:
        try:
            #eliminamos la base de datos
            client.drop_database(name)
            database_list = client.list_database_names()
            print("Nombres de bd:", database_list)
        except:
            print("Ha ocurrido algun error al borrar la bd")

    #creamos base de datos y las collecciones necesarios
    bdpractical = client["Practical"]
    database_list = client.list_database_names()
```

En la imagen anterior se puede apreciar que se han creado dos modos de funcionamiento para el programa, el modo 0 para la creación e inserción de datos y el modo 1 para la realización de peticiones para comprobar el funcionamiento del programa.

También se puede apreciar cómo se realiza la conexión con mongo en la nube y se pregunta al usuario si desea eliminar alguna bd, por si quiere crear las colecciones de 0.

```python
61      #añadir salario y comm atributo prescindible
62      #creo vectores con datos aleatorios
63      vName = ["I+D+I","Recursos Humanos","Relaciones Internacionales","Gestion Economica","Secretaria Tecnica","Marketing"]
64      vLoc = ["Espiel","Cordoba","Belmez","Pozoblanco","FuenteOvejuna","Villanueva del Rey"]
65      vEName = ["Antonio Gomez Gimenez", "Rafael Hormigo Cabello", "Alejandro Garcia Garcia", "Juanma Gomez Morales", "Imanol Rubio Plata",
66      "Clara Martinez Rodrigez", "Pilar Grande Romero", "Ana Garcia Fernandez", "Marta Diaz Pedrajas", "Lucia Ruiz Cid"]
67      vJOB = ["Informatico", "Electricista", "Contable", "Gestor", "Secretario", "Tecnico", "Fontanero", "Director"]
68      vSAL = [1000, 1500, 2000, 2500, 3000, 4000, 5000, 5500]
69      vHIREDATE = [datetime.strptime("2014-10-21", '%Y-%m-%d'),datetime.strptime("2017-9-13", '%Y-%m-%d'),datetime.strptime("2022-3-27", '%Y-
```

En la imagen anterior se pueden ver una serie de vectores que se han utilizado para la creación aleatoria de datos. Es decir, se crean unos vectores con datos predefinidos y aleatoriamente se accede a dichos datos.

```
71      if "Practical" in database_list:
72          print("La base de datos esta ya creada.")
73      else:
74          print("Creando BD...");
75
76      print("Creando Colecciones...");
77      database = client['Practical'];
78      #compruebo si existen la colecciones
79      collist = database.list_collection_names()
80
81      if "DEPT" in collist:
82          print("La coleccion DEPT existe.");
83          DEPTColeccion = database.get_collection("DEPT");
84      else:
85          #coleccion DEPT
86          DEPTColeccion = bdpractical.get_collection("DEPT")
87
88          ##################################################
89          # 3- Insertar datos en las colecciones.
90          ##################################################
91
92          for x in range(50):
93              mydict = {"_id": x, "DNAME": vName[random.randint(0, 5)], "LOC": vLoc[random.randint(0, 5)] }
94              DEPTColeccion.insert_one(mydict)
95
96          #--------------------------------------------------
```

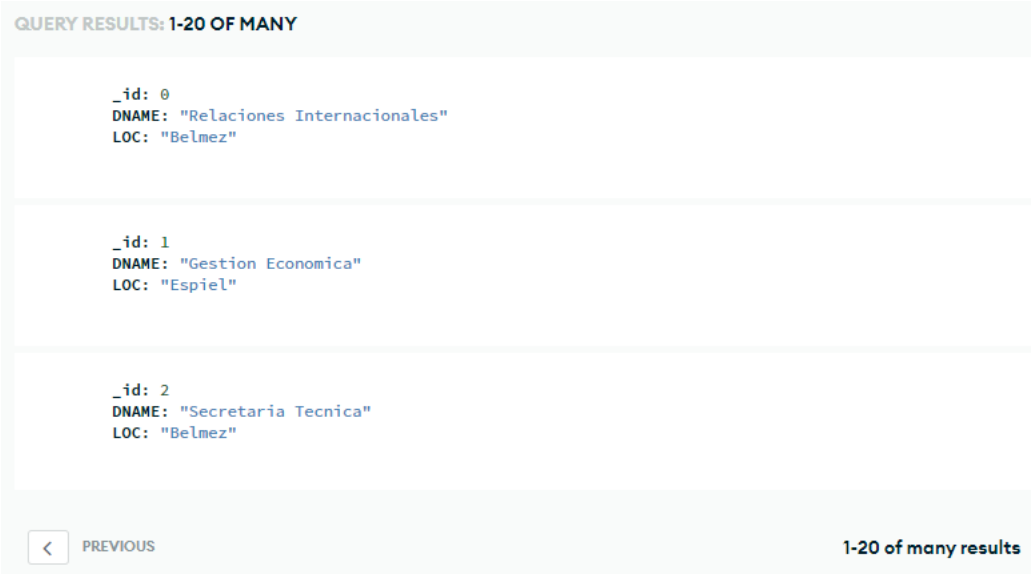
Primero se comprueba si la bd está ya creada y posteriormente se extrae la lista con las colecciones existentes en dicha bd. Posteriormente se comprueba si ya existía en la lista la colección DEPT. Si existe se extrae la colección, sino, se crea la colección con los vectores aleatorios de datos.

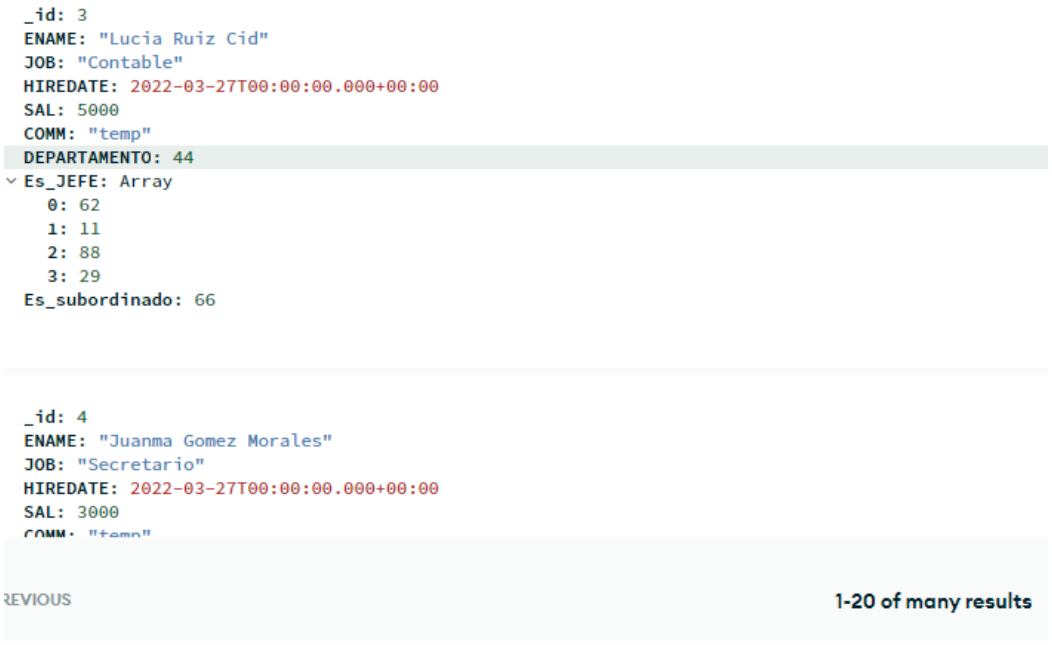
```
 98      if "EMP" in collist:
 99          print("La coleccion EMP existe.");
100          EMPColeccion = database.get_collection("EMP");
101      else:
102          #coleccion EMP
103          bdpractical.create_collection("EMP")
104          EMPColeccion = bdpractical.get_collection("EMP")
105
106          ##################################################
107          # 3- Insertar datos en las colecciones.
108          ##################################################
109          for x in range(100):
110              #referencia cambiada por id de la collecion referida #{"$ref": "DEPT", "$_id": random.randint(0,
111              mydict = {"_id": x, "ENAME": vEName[random.randint(0, 9)], "JOB": vJOB[random.randint(0, 7)], "HI
112              EMPColeccion.insert_one(mydict)
113
114          for x in range(100):
115              #referencias cambiada por id de la collecion referida #{"$ref": "EMP", "$_id": valoraleatoriox}
116              query = {"_id": x}
117              valoraleatorio = random.randint(0, 99)#para subordinado
118              valoraleatorio2 = random.randint(0, 99)#para jefe
119              #
120               #valor aleatorio para ver cuantos subordinados tiene
121              valoraleatorio3 = random.randint(1, 5)
122              vecSubordinados = []
123              for w in range(valoraleatorio3):
124                  valoraleatorio4 = random.randint(0, 99)
125                  while valoraleatorio4 == x:#compruebo que no sea el mismo
126                      valoraleatorio4 = random.randint(0, 99)
127                  vecSubordinados.append(valoraleatorio4)
128
129              if valoraleatorio == x: ##controlo que no pueda ser jefe o subordinado de si mismo
130                  if valoraleatorio2 == x:
131                      mydict = {"$set": {"Es_JEFE": "", "Es_subordinado": ""}}
132                  else:
133                      mydict = {"$set": {"Es_JEFE": valoraleatorio2, "Es_subordinado": ""}}
134              else:
135                  if valoraleatorio2 == x:
136                      mydict = {"$set": {"Es_JEFE": "", "Es_subordinado": valoraleatorio2}}
137                  else:
138                      mydict = {"$set": {"Es_JEFE": vecSubordinados, "Es_subordinado": valoraleatorio2}}
139
140              EMPColeccion.update_one(query, mydict)
```

En la imagen anterior el concepto es similar a la imagen de creación de la colección DEPT pero para la colección EMP. La diferencia es que para asignar los id que hacen referencia tanto a otros empleados como el departamento hay que tener en cuenta que no se repita el mismo id para el mismo empleado y que un empleado puede ser jefe de 0 o n empleados. Teniendo todo esto en cuenta, se creó el código anterior.

El resto del código hace referencia al apartado siguiente.

Algunos ejemplos de datos para las dos colecciones son los siguiente:





2. Apartado 3 - Consultas a la información:

En este apartado, se creó el modo 1 para realizar distintas consultas y comprobar los resultados para ver si la bd funcionaba correctamente. Las consultas son las siguientes:

```
151     if "Practical" in database_list:
152         database = client['Practical'];
153         #compruebo si existen la colecciones
154         collist = database.list_collection_names()
155         if "DEPT" in collist and "EMP" in collist:
156 
157             bdpractical = client["Practical"]
158             DEPTColeccion = bdpractical.get_collection("DEPT")
159             EMPColeccion = bdpractical.get_collection("EMP")
160 
161             ##peticion usando find_one
162             print('-----------------------------------------------')#peticion para primer
163             print(DEPTColeccion.find_one( {}, { "DNAME": 1, "LOC" : 1}))
164             print('-----------------------------------------------')#peticion que devuelv
165             print(DEPTColeccion.find_one( {"DNAME" : "Recursos Humanos"}, {"LOC" : 0} ))
166             print('-----------------------------------------------')#peticion que devuelv
167             print(DEPTColeccion.find_one( {"_id" : 47}, {"LOC" : 1} ))
168             print('-----------------------------------------------')
```

Primeramente se realizaron tres peticiones usando find_one, donde la primera petición nos da el primer encuentro. La segunda petición devuelve el primer encuentro con recursos humanos y devuelve todo menos la localidad. La tercera petición devuelve el primer encuentro con el id 47 dando la localidad. Los resultados son los siguientes:

```
antoniogg@antoniogg-SATELLITE-L50D-C:~/Escritorio/adp/Practica1$ python3.8 ./Pra
ctica1.py -m1
Consultas a la informacion
-----------------------------------------------
{'_id': 0, 'DNAME': 'Relaciones Internacionales', 'LOC': 'Belmez'}
-----------------------------------------------
{'_id': 3, 'DNAME': 'Recursos Humanos'}
-----------------------------------------------
{'_id': 47, 'LOC': 'FuenteOvejuna'}
-----------------------------------------------
```

Posteriormente se realizaron las siguientes consultas:

```
170             #consulta que devuelve los resultados que coinciden con el id 24 o 11
171             result_1 = DEPTColeccion.find({"_id" : { "$in" : [ 24, 11] }})
172             for i in result_1:
173                 print(i)
174             print('-----------------------------------------------')
175 
176             #consulta que devuelve los resultados cuyo Localidad es Espiel y su id esta entre 40 y 10
177             result_1 = DEPTColeccion.find({"LOC" : "Espiel","_id" : { "$lt" : 40, "$gt" : 10}})
178             for i in result_1:
179                 print(i)
180             print('-----------------------------------------------')
```

La primera devuelve los resultados que coinciden con el id 24 o 11 y la segunda, devuelve los resultados cuyo localidad es Espiel y su id está entre 40 y 10. Los resultados son los siguientes:

```
------------------------------------------------
{'_id': 11, 'DNAME': 'Secretaria Tecnica', 'LOC': 'Villanueva del Rey'}
{'_id': 24, 'DNAME': 'Relaciones Internacionales', 'LOC': 'Villanueva del Rey'}
------------------------------------------------
{'_id': 12, 'DNAME': 'Recursos Humanos', 'LOC': 'Espiel'}
{'_id': 14, 'DNAME': 'Secretaria Tecnica', 'LOC': 'Espiel'}
{'_id': 18, 'DNAME': 'Marketing', 'LOC': 'Espiel'}
{'_id': 20, 'DNAME': 'Recursos Humanos', 'LOC': 'Espiel'}
{'_id': 21, 'DNAME': 'Relaciones Internacionales', 'LOC': 'Espiel'}
{'_id': 25, 'DNAME': 'I+D+I', 'LOC': 'Espiel'}
------------------------------------------------
```

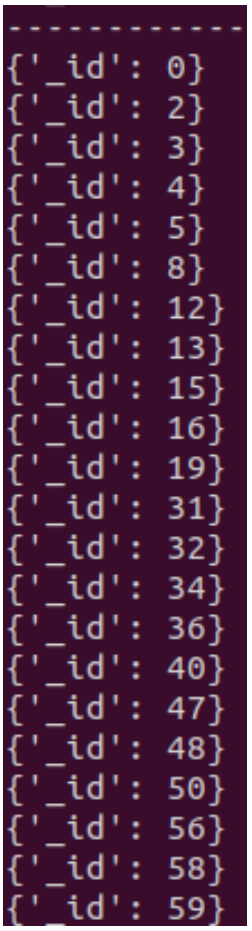
Posteriormente se realizaron las siguientes consultas:

```
182             #consulta que devuelve los resultados cuyo nombre es Relacione
183             result_1 = DEPTColeccion.find({
184             "$and" : [{
185                             "DNAME" : { "$eq" : "Relaciones Internacionales"}
186                         },
187                         {
188                             "LOC" : { "$ne" : "Espiel"}
189                         }]
190             })
191             for i in result_1:
192                 print(i)
193             print('------------------------------------------------')
194
195             #consulta que devuelve de la coleccion empleado las personas q
196             result_1 = EMPColeccion.find({
197             "$or" : [{
198                             "ENAME" : { "$eq" : "Juanma Gomez Morales"}
199                         },
200                         {
201                             "JOB" : { "$in" : ["Gestor", "Contable"]}
202                         }]
203             },{"_id": 1})
204
205             for i in result_1:
206                 print(i)
207             print('------------------------------------------------')
```

La primera devuelve los resultados cuyo nombre es Relaciones Internacionales y no se encuentran en Espiel y la segunda, devuelve de la colección empleados, las personas que se llaman Juanma o son gestores o contables y devuelvo solo el id. Los resultados son los siguientes:

```
------------------------------------------------
{'_id': 0, 'DNAME': 'Relaciones Internacionales', 'LOC': 'Belmez'}
{'_id': 15, 'DNAME': 'Relaciones Internacionales', 'LOC': 'Cordoba'}
{'_id': 24, 'DNAME': 'Relaciones Internacionales', 'LOC': 'Villanueva del Rey'}
{'_id': 41, 'DNAME': 'Relaciones Internacionales', 'LOC': 'Belmez'}
{'_id': 49, 'DNAME': 'Relaciones Internacionales', 'LOC': 'Pozoblanco'}
------------------------------------------------
```

Para la primera colección.

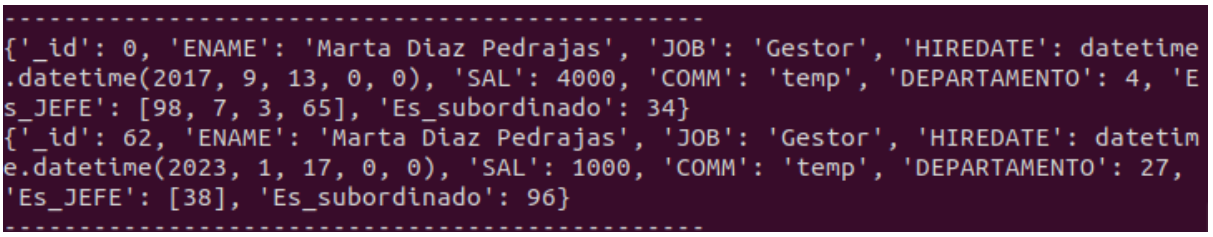


Para la segunda colección, cabe destacar que hay aún más id que coinciden con esta búsqueda.

Finalmente, se realizó la última consulta:

```
207         print('-----------------------------------------------')
208
209         #consulta que devuelve la informacion de todas las persona
210         result_1 = EMPColeccion.find({
211         "JOB" : { "$regex" : "^G" },
212         "ENAME": "Marta Diaz Pedrajas"
213         })
214
215         for i in result_1:
216             print(i)
217         print('-----------------------------------------------')
```

En esta consulta se busca la información de todas las personas que se llaman Marta Díaz Pedrajas y su trabajo comienza por G. El resultado es el siguiente:



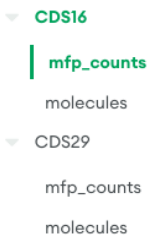
FASE 2: APLICACIONES CIENTÍFICAS Y EMPRESARIALES (I)

Para la realización de dicha práctica se ha dividido el trabajo en dos partes. La primera parte realiza la obtención de los datos de la bd que se encuentra en la nube de Atlas y realiza el procesamiento necesario para la creación de un fichero de texto con los datos necesarios, y la segunda parte, realiza toda la parte relacionada con la creación y desarrollo del modelo.

El objetivo de dicha práctica es utilizar las bases de datos de compuestos del ejemplo práctico desarrollado en la parte teórica de la asignatura, implementar el código python necesario para predecir(clasificación) el tipo de actividad biológica (Active/Inactive).

2.1. Parte 1:

Para la realización de la primera parte, primeramente se han subido a mongo Atlas las dos bases de datos de prueba. En la siguiente imagen se pueden apreciar las dos bases de datos:



Una vez realizada la subida de los datos a la nube, es necesario extraer los datos de la bd que vamos a utilizar, para ello se creó el fichero GenerarCsv.py que genera un fichero donde se encuentran los datos a utilizar. Principalmente se crea un fichero Dataset.csv porque a la hora de trabajar sobre los datos de la bd, los tiempos de trabajo son demasiado grandes, aparte es necesario que la matriz con la estructura de los datos sea int8, ya que sino, la memoria del programa aumenta de tal forma que el proceso es parado por el propio sistema operativo.

El código es el siguiente:

```
##################################################
# 1- Importar las bibliotecas y realizar la conexión
##################################################

# importing the required libraries
import pymongo
import json
import warnings
import pandas as pd
import numpy as np
warnings.filterwarnings('ignore')

#########################################################################################################
#Detalles conexion
connection_details  = 'mongodb+srv://AntonioGG:patata@cluster0.uoqzgcw.mongodb.net/?retryWrites=true&w=majority'

#Detalles_2 base de datos molecular
#db_name='CDS16';
db_name='CDS29';

# connect to the mongoclient
client = pymongo.MongoClient(connection_details);

# the list_database_names() method returns a list of strings
database_names = client.list_database_names();

if db_name in database_names:
  print('The database ' +  db_name + ' exists');
  db=client.get_database(db_name);
else:
    print('&&&&&& The database ' +  db_name + ' must be loaded &&&&&.');
    print()
    sys.exit()

npatrones = db.molecules.count_documents({})
print(npatrones)

pdEntrada = pd.DataFrame(np.zeros((npatrones, 1024), dtype=np.int8))
pdSalida = pd.DataFrame(np.zeros((npatrones,1),dtype=np.int8))
```

En la imagen anterior se realiza la conexión con la bd y la creación de dos matrices donde se almacenará un matriz de input para el modelo y de outputs. Siendo los inputs una matriz de tamaño, la cantidad de documentos por 1024, con valores 0 o 1, dependiendo de la molécula. Respecto a los outputs, es una columna de tamaño, la cantidad de documentos, almacenando 0 si la molécula es inactiva y 1 si es activa.

Para la creación del Dataset final que se almacena y la creación de dichas matrices, se realizó el siguiente código:

```
41  print(pdEntrada.shape)
42
43  iterador = 0
44
45  result = db.molecules.find({},{"_id": 0, "mfp.bits": 1, "class":1})
46
47  for i in result:
48    for posicion in i['mfp']['bits']:
49      pdEntrada[posicion][iterador] = 1
50    if i['class'] == "Active":
51      pdSalida[0][iterador] = 1
52    iterador = iterador + 1
53  result = []
54
55  dataset = pd.concat([pdEntrada, pdSalida], axis=1)
56
57  dataset.to_csv('Dataset.csv',index=False)
58  #eliminar linea de index que se genera
59
60  client.close();
```

Como podemos ver en la imagen, se realiza la petición find y sobre la propia petición, se almacenan los 1 en los lugares donde se nos indique en la petición.
Una vez tenemos las dos matrices, se concatenan en una sola y se guardan en Dataset.csv.

Cabe destacar que en dicho fichero se genera una línea de índice, es necesario eliminarla para que en el fichero encargado del modelo funcione correctamente.

2.2. Parte 2:

Para la realización de la segunda parte, se creó el fichero modelo.py, que almacena los mejores modelos entrenados, genera unos ficheros de información de cada modelo durante el entrenamiento con los resultados de cada combinación de parámetros, y comenta cual es el mejor modelo para el fichero Dataset.csv.

Para la realización del código, se decidió utilizar tres tipos de modelos:

- **SVM**. Máquinas de Vectores soporte, muy buenos para problemas de clasificación de solo dos tipos de resultado, en nuestro caso, si o no.
- **Árboles de decisión**. Buenos para problemas de clasificación de dos clases o más, se decidió usar dicho modelo por su fácil entendimiento.
- **Random Forest.** Un ensemble de árboles de decisión, se introdujo este modelo para comprobar si, a mayor número de árboles, se realiza una mejora sobre el modelo resultante.

Para comprobar el entrenamiento de los modelos y la validez de los mismos, se usó el CCR (Correct Classification Rate), que nos da como valor, cuantos valores ha predicho correctamente el modelo frente a los que realmente son.

Para poder validar los modelos es necesario que dichos datos no se utilicen en el entrenamiento, ya que sería algo similar a hacer trampa, que el propio modelo conoce los resultados que deberían dar al haber entrenado con dichos datos.

Para dar mayor robustez a los modelos se realizó un k-fold de 5 para permitir entrenar con 5 conjuntos de datos distintos y almacenar la media del conjunto.

Para la realización de dicho código, se realizaron las siguientes funciones de ayuda:

Función read_data:

```
21  ##############################################################################
22  #funcion para leer datos del fichero
23  ##############################################################################
24  def read_data(fichero):
25
26      input = None
27      output = None
28      train_inputs = None
29      train_outputs = None
30      test_inputs = None
31      test_outputs = None
32
33      #guardamos los datos en un dataset de tipo string
34      data = pd.read_csv(fichero)
35      input = data.iloc[:,:-1].values
36      output = data.iloc[:,-1].values
37
38      #se juntan los dos dataset
39      #Dataset = pd.concat([Dataset_train, Dataset_test], axis=1)
40
41      #dividimos en 80% de datos para entreno y el resto para test
42      train_inputs, test_inputs, train_outputs, test_outputs = train_test_split(input, output, test_size= 0.20, stratify=output)
43
44      return train_inputs, train_outputs, test_inputs, test_outputs, input, output
```

La función anterior, dado un fichero, realiza la lectura del mismo y la división de los datos en un conjunto de train y test (con sus respectivos inputs y outputs). Cabe destacar, que la división de los datos se realiza de forma estratificada para evitar que algún conjunto sólo esté formado por valores activos o inactivos (y decaiga la eficiencia del modelo a entrenar o la validación del mismo).

Función guardar_modelo:

```
47  ###############################################################################
48  #funcion para guardar un modelo
49  ###############################################################################
50  def guardar_modelo(modelo, name):
51
52      #guardamos el modelo y el scaler
53      with open(name+'.pickle', 'wb') as fw:
54          pickle.dump(modelo, fw)
```

Función sencilla que dado un modelo y un nombre para guardarlo, guarda el modelo en formato .pickle (por si se desea comprobar el funcionamiento del modelo).

Función entrenarSVR:

```
57  ###############################################################################
58  #funcion para entrenar SVR
59  ###############################################################################
60  def entrenarSVR(train_inputs, train_outputs, test_inputs, test_outputs, input, output):
61
62      f=open("ResultadosParametrosSVR", "w"); ##archivo donde se almacenan los resultados de los parametros
63
64      #inicializamos parametros que usamos en el entrenamiento
65      vector = np.array([1e-1,1e0,1e1,1e2])
66      mejor_mse = 999999
67      mejor_C = 999999
68      mejor_Gamma = 999999
69
70      skf= StratifiedKFold(n_splits=5)
71      mejorconf=0
72      for C in vector:#entrenamos probando distintas c y gammas
73          for Gamma in vector:
74              contador = 0
75              media = 0
76              print("C=%f y Gamma=%f" % (C, Gamma))
77
78              for train_index, test_index in skf.split(train_inputs, test_inputs):
79                  entrada_train, entrada_test = input[train_index], input[test_index]
80                  salida_train, salida_test = output[train_index], output[test_index]
81
82                  # Entrenamos el modelo
83                  modelo = svm.SVC(kernel='rbf',C=C, gamma=Gamma)
84                  modelo.fit(entrada_train, salida_train)
85                  y_pred=modelo.predict(entrada_test)
86                  test_ccr = (precision_score(salida_test, y_pred, average='micro'))*100
87                  contador = contador + test_ccr
88              media = contador/5
89
90              f.write("CCR Final con C=%f y G=%f: \t%f\n" % (C, Gamma, media))#lo vamos almacenando en un fichero
91
92              if media > mejorconf:#se va guardando la mejor combinacion de parametros
93                  mejorconf=media
94                  mejor_C = C
95                  mejor_Gamma = Gamma
```

```
96
97      #Se realiza la validación del modelo
98      modelo = svm.SVC(kernel='rbf',C=mejor_C, gamma=mejor_Gamma)
99      modelo.fit(train_inputs, test_inputs)
100     y_pred = modelo.predict(train_outputs)
101     test_ccr = (precision_score(test_outputs, y_pred, average='micro'))*100
102
103     #se almacena el modelo y el scaler
104     guardar_modelo(modelo,"SVR")
105
106     f.write("Los mejores parametros son C:"+ str(mejor_C) +" y Gamma:"+ str(mejor_Gamma) +" con un CCR de "+ str(test_ccr))
107     f.close()
108
109     print("*****************")
110     print("Los mejores parametros son C:"+ str(mejor_C) +" y Gamma:"+ str(mejor_Gamma) +" con un CCR de "+ str(test_ccr))
111     print("*****************")
112
113     return test_ccr
```

Esta función es similar a entrenarArbolDecision y entrenarRandomForest, ya que todas tienen la misma estructura pero con la diferencia de los parámetros de las funciones que utilizan los diferentes modelos.

Como podemos ver en las imágenes anteriores, se realiza la configuración de parámetros que se van a probar para la función del modelo, y se realiza un k-fold de 5 para dar mayor robustez. Para cada caso se realiza el entrenamiento, predicción y comprobar si ha sido el que ha dado el mejor resultado. Una vez se realizan todas las pruebas, se comprueba la validez del resultado que ha dado mejor CCR con los datos de test. Finalmente, se devuelve dicho resultado.
Cabe destacar, que en paralelo se van almacenando en un fichero los resultados de cada prueba y del mejor valor.

Función entrenarArbolDecision:

```
116 ##############################################################################
117 #funcion que entrena con el modelo ArbolDecision
118 ##############################################################################
119 def entrenarArbolDecision(train_inputs, train_outputs, test_inputs, test_outputs, input, output):
120
121     f=open("ResultadosParametrosArbolDecision", "w"); ##archivo donde se almacenan los resultados de los parametros
122
123     #inicializamos parametros que usamos en el entrenamiento
124     v_splitter = {"best","random"}
125     v_min_samples_split = {0.1, 0.3, 0.5, 0.6, 0.8}
126     v_min_samples_leaf = {0.1, 0.3, 0.5}
127     mejor_mse = 99999
128     mejor_splitter = None
129     mejor_min_samples_split = 999999
130     mejor_min_samples_leaf = 999999
131
132     skf= StratifiedKFold(n_splits=5)
133     mejorconf=0
134     #entrenamos probando distintas combinaciones
135     for splitter in v_splitter:
136         for min_samples_split in v_min_samples_split:
137             for min_samples_leaf in v_min_samples_leaf:
138                 contador = 0
139                 media = 0
140                 print("splitter=%s, min_samples_split=%f, min_samples_leaf=%f\n" % (splitter, min_samples_split, min_samples_leaf))
141
142                 for train_index, test_index in skf.split(train_inputs, test_inputs):
143                     entrada_train, entrada_test = input[train_index], input[test_index]
144                     salida_train, salida_test = output[train_index], output[test_index]
145
146                     #Entrenamos el modelo
147                     modelo = tree.DecisionTreeClassifier(splitter=splitter, min_samples_split=min_samples_split, min_samples_leaf=min_samples_leaf)
148                     modelo.fit(entrada_train, salida_train)
149                     y_pred=modelo.predict(entrada_test)
150                     test_ccr = (precision_score(salida_test, y_pred, average='micro'))*100
151                     contador = contador + test_ccr
152                 media = contador/5
```

```
                f.write("CCR Final con splitter=%s, min_samples_split=%f y min_samples_leaf=%f: \t%f\n" % (splitter, min_samples_split, min_samples_leaf, media))

                if media > mejorconf:#se va guardando la mejor combinacion de parametros
                    mejorconf=media
                    mejor_splitter = splitter
                    mejor_min_samples_split = min_samples_split
                    mejor_min_samples_leaf = min_samples_leaf


    #Se realiza la validación del modelo
    modelo = tree.DecisionTreeClassifier(splitter=mejor_splitter, min_samples_split=mejor_min_samples_split, min_samples_leaf=mejor_min_samples_leaf)
    modelo.fit(train_inputs, test_inputs)
    y_pred = modelo.predict(train_outputs)
    test_ccr = (precision_score(test_outputs, y_pred, average='micro'))*100

    #se almacena el modelo
    guardar_modelo(modelo,"ArbolDecision")

    f.write("Los mejores parametros para el arbol de decision es splitter->"+ str(mejor_splitter)+", min_samples_split->"+ str(mejor_min_samples_split)+", min_sam
    f.close()

    print("******************")
    print("Los mejores parametros para el arbol de decision es splitter->"+ str(mejor_splitter)+", min_samples_split->"+ str(mejor_min_samples_split)+", min_sampl
    print("******************")

    return test_ccr
```

Similar al caso anterior.

Función entrenarRandomForest:

```
#############################################################################
#funcion que entrena con el modelo RandomForest
#############################################################################
def entrenarRandomForest(train_inputs, train_outputs, test_inputs, test_outputs, input, output):

    f=open("ResultadosParametrosRandomForest", "w"); ##archivo donde se almacenan los resultados de los parametros

    #inicializamos parametros que usamos en el entrenamiento
    v_n_estimators = {100,300}
    v_max_features = {0.01, 0.1,0.3}
    v_min_samples_split = {2, 50, 100}
    v_min_samples_leaf = {2, 50, 100}
    mejor_mse = 99999
    mejor_n_estimators = 999999
    mejor_min_samples_split = 999999
    mejor_min_samples_leaf = 999999
    mejor_max_features = 999999

    skf= StratifiedKFold(n_splits=5)
    mejorconf=0
    #entrenamos probando distintas combinaciones
    for n_estimators in v_n_estimators:
        for max_features in v_max_features:
            for min_samples_split in v_min_samples_split:
                for min_samples_leaf in v_min_samples_leaf:
                    contador = 0
                    media = 0
                    print("n_estimators=%d, max_features=%f, min_samples_split=%f, min_samples_leaf=%f \n" % (n_estimators

                    for train_index, test_index in skf.split(train_inputs, test_inputs):
                        entrada_train, entrada_test = input[train_index], input[test_index]
                        salida_train, salida_test = output[train_index], output[test_index]

                        #Entrenamos el modelo
                        modelo = ensemble.RandomForestClassifier(min_weight_fraction_leaf=0.2, max_samples= 0.5,n_estimato
                        modelo.fit(entrada_train, salida_train)
                        y_pred=modelo.predict(entrada_test)
                        test_ccr = (precision_score(salida_test, y_pred, average='micro'))*100
                        contador = contador + test_ccr
                    media = contador/5

                    f.write("CCR Final con n_estimators=%d, v_max_features=%f, min_samples_split=%f, min_samples_leaf=%f=
```

```
                    if media > mejorconf:#se va guardando la mejor combinacion de parametros
                        mejorconf=media
                        mejor_n_estimators = n_estimators
                        mejor_min_samples_split = min_samples_split
                        mejor_min_samples_leaf = min_samples_leaf
                        mejor_max_features = max_features


        #Se realiza la validación del modelo
        modelo = ensemble.RandomForestClassifier(min_weight_fraction_leaf=0.2, max_samples= 0.5, n_estimators=mejor_n_estimators, m
        modelo.fit(train_inputs, test_inputs)
        y_pred = modelo.predict(train_outputs)
        test_ccr = (precision_score(test_outputs, y_pred, average='micro'))*100

        #se almacena el modelo
        guardar_modelo(modelo,"RandomForest")

        f.write("Los mejores parametros para el random forest es n_estimators->"+ str(mejor_n_estimators) + "max_features->"+ str(m
        f.close()

        print("******************")
        print("Los mejores parametros para el random forest es n_estimators->"+ str(mejor_n_estimators) + "max_features->"+ str(mej
        print("******************")

        return test_ccr
```

Similar al caso anterior.

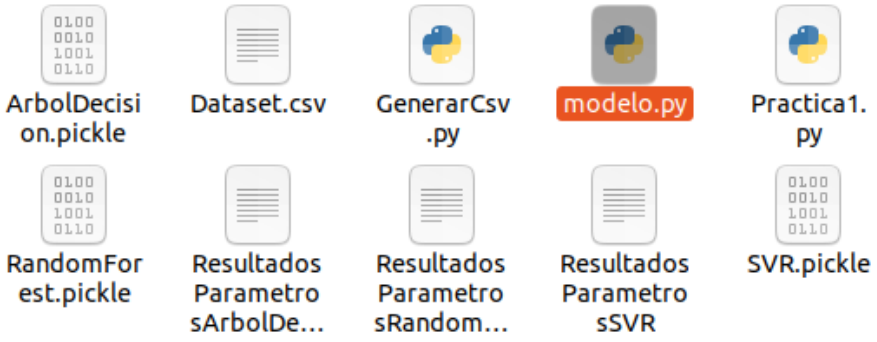
Código principal:

```
#############################################################################
#funcion principal
#############################################################################

#nombre de los ficheros
fichero = "Dataset.csv"

#llamamos a funcion read_data para extraer los datos del fichero csv a un formato con el que podamos trabajar
train_inputs, train_outputs, test_inputs, test_outputs, input, output= read_data(fichero,)
if train_inputs is None or train_outputs is None or test_inputs is None or test_outputs is None or input is None or output is None:
    print("Error con el fichero")
    exit()

#No se aplica standarscaler ya que los datos son 0 y 1, siendo datos muy sencillos

#creamos una lista con el nombre de todos los modelos
NombreModelo = ["SVM", "ArbolDecision", "RandomForest"]
listaCCR = np.zeros(shape=(3), dtype=float)

#entrenamos con los distintos modelo para ver cual es el mejor
print("Entrenando " + NombreModelo[0])
listaCCR[0] = entrenarSVR(train_inputs, test_inputs, train_outputs, test_outputs, input, output)
print("Entrenando " + NombreModelo[1])
listaCCR[1] = entrenarArbolDecision(train_inputs, test_inputs, train_outputs, test_outputs, input, output)
print("Entrenando " + NombreModelo[2])
listaCCR[2] = entrenarRandomForest(train_inputs, test_inputs, train_outputs, test_outputs, input, output)

mejorModelo = -1
mejorCCR = 0

for iterador in range(len(listaCCR)):#comprobamos cual genero un CCR mayor
    if listaCCR[iterador] > mejorCCR:
        mejorCCR = listaCCR[iterador]
        mejorModelo = iterador

print("El mejor modelo es " + NombreModelo[mejorModelo] + " con un CCR de: " + str(mejorCCR))
```

El código principal, simplemente va llamando a las funciones creadas anteriormente y muestra el como resultado, aquel modelo que ha proporcionado un CCR con el mayor valor.

Para comprobar el funcionamiento de dicho programa, se muestran a continuación los ficheros generados y el resultado obtenido.

Los fichero generados son los siguientes:



Como podemos observar, se han generado los ficheros .pickle que almacenan los tres modelos y los ficheros con los resultados de las distintas configuraciones.

Y respecto al resultado del programa:

```
******************
El mejor modelo es SVM con un CCR de: 80.3991446899501
```

Si quisiéramos obtener mayor información del modelo, por ejemplo, SVM, miraríamos el fichero ResultadosParametrosSVR donde encontraríamos:

```
1  CCR Final con C=0.100000 y G=0.100000:  76.382938
2  CCR Final con C=0.100000 y G=1.000000:  76.382938
3  CCR Final con C=0.100000 y G=10.000000:     76.382938
4  CCR Final con C=0.100000 y G=100.000000:    76.382938
5  CCR Final con C=1.000000 y G=0.100000:  76.151241
6  CCR Final con C=1.000000 y G=1.000000:  76.418588
7  CCR Final con C=1.000000 y G=10.000000:     76.418588
8  CCR Final con C=1.000000 y G=100.000000:    76.418588
9  CCR Final con C=10.000000 y G=0.100000:     75.153388
10 CCR Final con C=10.000000 y G=1.000000:     76.400779
11 CCR Final con C=10.000000 y G=10.000000:    76.418588
12 CCR Final con C=10.000000 y G=100.000000:   76.418588
13 CCR Final con C=100.000000 y G=0.100000:    75.153388
14 CCR Final con C=100.000000 y G=1.000000:    76.400779
15 CCR Final con C=100.000000 y G=10.000000:   76.418588
16 CCR Final con C=100.000000 y G=100.000000:  76.418588
17 Los mejores parametros son C:1.0 y Gamma:1.0 con un CCR de 80.3991446899501
```

Como podemos ver, se nos muestran los resultados obtenidos para cada combinación de parámetros, y el resultado de la mejor combinación probando su funcionamiento con los datos de test en la validación.

Tanto entrenarArbolDecision como entrenarRandomForest, generaban el mismo CCR de 76.3829. Esto se debe a que daban igual la configuración de parámetros introducidos al modelo, generaban los mismo resultados. Es decir, que un árbol de decisión para este dataset de datos generaba los mismos resultados y el ensemble al ser una agrupación de los mismos, arrastraba dicho problema.

Por lo tanto para este Dataset, no se recomienda el uso de estos dos modelos, se recomienda mejor el modelo SVR que genera un resultado mejor para este Dataset.

Cabe destacar que se intentó realizar un plot sobre svm. Pero al tener 1024 dimensiones de entrada sería difícil comprobar su funcionamiento, se suele usar estos plot para bd sintéticas con dos parámetros de entrada permitiendo observar en un plano 2d como se genera la línea que diferencia las dos clases. Ya que al incrementar el número de parámetros de entrada, entran en juego mayor número de planos. Si acaso se podría haber realizado un plot que compare los resultados generados por el modelo (svr ya que los otros carece de sentido) frente a los resultados reales por el test, pero con el propio CCR ya obtenemos el porcentaje de acierto del modelo, como mucho serviría para ver dónde ha fallado exactamente.

Como breve conclusión tras lanzar este programa, podemos decir que para el Dataset obtenido de la bd, podemos conseguir un modelo SVR con una tasa de acierto del 80.39%, teniendo en cuenta que con una tasa de entrenamiento mayor (mayor cantidad de pruebas y parámetros) o diferentes modelos, se podría obtener un resultado mayor, lógicamente sin llegar al sobre-entrenamiento.