UNIVERSIDAD
D
CÓRDOBA

Instituto de Estudios de Posgrado
Universidad de Córdoba

Máster Universitario en Inteligencia Computacional e Internet de las Cosas

Solución NoSQL MongoDB. Caracterización y Predicción de Actividades/Propiedades de Compuestos (Diseño de fármacos)

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

Autora:
Alba Márquez-Rodríguez

Profesores:
Gonzalo Cerruela García
Dormingo Ortiz Boyer
Juan A. Romero del Castillo

Córdoba, Diciembre 2023

Índice

1. Enunciado

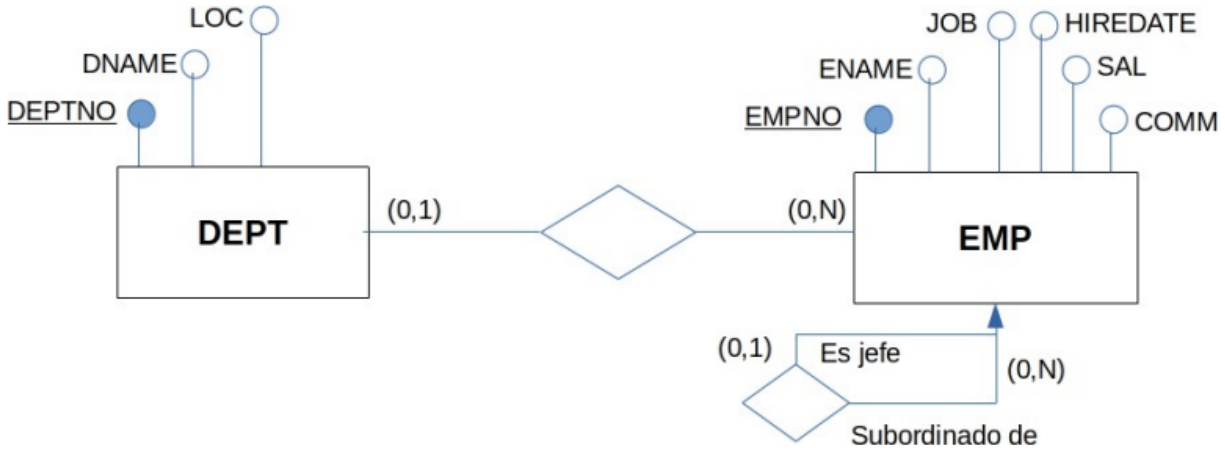
1.1. Fase 1: Solución NoSQL MongoDB

1.1.1. Objetivo

El objetivo de esta práctica es afianzar los conocimientos impartidos en la parte teórica de la asignatura sobre el uso de la solución NoSQL MongoDB.

1.1.2. Trabajo a Realizar

A partir del siguiente esquema entidad-interrelación:



1. Proponga una estructura de colecciones de documentos MongoDB para almacenar la información.
2. Realizar una carga de información lo suficientemente completa para que le sirva como prueba y validación.
3. Realice varias consultas generando informes de salida que le permitan comprobar el acceso a la información.

1.2. Fase 2: Aplicaciones científicas y empresariales

1.2.1. Objetivo

El objetivo de esta práctica es afianzar los conocimientos impartidos en la parte teórica de la asignatura sobre la caracterización y predicción de actividades/propiedades de compuestos.

1.2.2. Trabajo a Realizar

Utilizando 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).

Recomendaciones para la implementación:

1. Utilice alguno de los modelos de clasificación implementados en la biblioteca scikit-learn.
2. El tipo de actividad de cada compuesto (variable respuesta) está almacenado en el campo "class" de la colección "molecules".
3. Como variables predictoras se utilizará el fingerprint molecular de cada compuesto. Recuerde que en la base de datos sólo están almacenadas las posiciones de los bits con valor "1".

   Las variables predictoras y la respuesta quedarían representadas de siguiente forma:

id_Compuesto	FP1	FP2	...	FP1024	class
comp-1	0	0		1	Active(1)
comp-2	1	0		1	Inactive(0)
comp-N	1	0		0	Active(1)

4. Las columnas donde todos los valores sean '0' pueden ser eliminadas.
5. Para evaluar el desempeño de los clasificadores se deben utilizar métricas clásicas como GMean, Kappa, Accuracy, AURoc, etc.

2. Fase 1: Solución NoSQL MongoDB

El esquema entidad-relación que muestra dos entidades:

- DEPT (departamento)
- EMP (empleado)

Estas entidades tienen atributos propios y una relación que indica que un empleado puede ser jefe de un departamento y a su vez puede tener subordinados.

2.1. Estructura de Colecciones MongoDB

Para diseñar una estructura de colecciones de documentos MongoDB basada en este esquema, se puede seguir el siguiente enfoque:
Se podrían tener dos colecciones principales, una para cada entidad:

- Departamentos
- Empleados

2.1.1. Colección Departamentos (Departments):

```
{
  "_id": ObjectId("..."),
  "DEPTNO": "N mero del departamento",
  "DNAME": "Nombre del departamento",
  "LOC": "Localizaci n del departamento",
  "MANAGER": "N mero del empleado (EMPNO) que es jefe del departamento",
}
```

2.1.2. Colección Empleados (Employees):

```
{
  "_id": ObjectId("..."),
  "EMPNO": "N mero del empleado",
  "ENAME": "Nombre del empleado",
  "JOB": "Trabajo del empleado",
  "HIREDATE": "Fecha de contrataci n del empleado",
  "SAL": "Salario del empleado",
  "COMM": "Comisi n del empleado",
  "DEPTNO": "N mero del departamento al que pertenece el empleado",
  "ISBOSS": "Es jefe",
  "BOSS": "Si no es jefe, de qui n es subordinado"
  ]
}
```

2.1.3. Creación de la Base de Datos y Colecciones en MongoDB

Para la creación de la Base de Datos, Colecciones y datos se ha utilizado python.

Primero se realiza la conexión:

```
# connect to the mongoclient
client = pymongo.MongoClient(MONGO_LINK)

client = pymongo.MongoClient('mongodb://localhost:27017')
```

Luego, se crea la base de datos, que se llama `HRManagement`. En el caso de que exista, accederá a ella, si no existe, la crea:

```
# get the database
database = client['HRManagement']
```

A continuación, se crean las colecciones `Departments` y `Employees`:

```
collections = ['Departments', 'Employees']
for collection_name in collections:
    if collection_name not in database.list_collection_names():
        database.create_collection(collection_name)
```

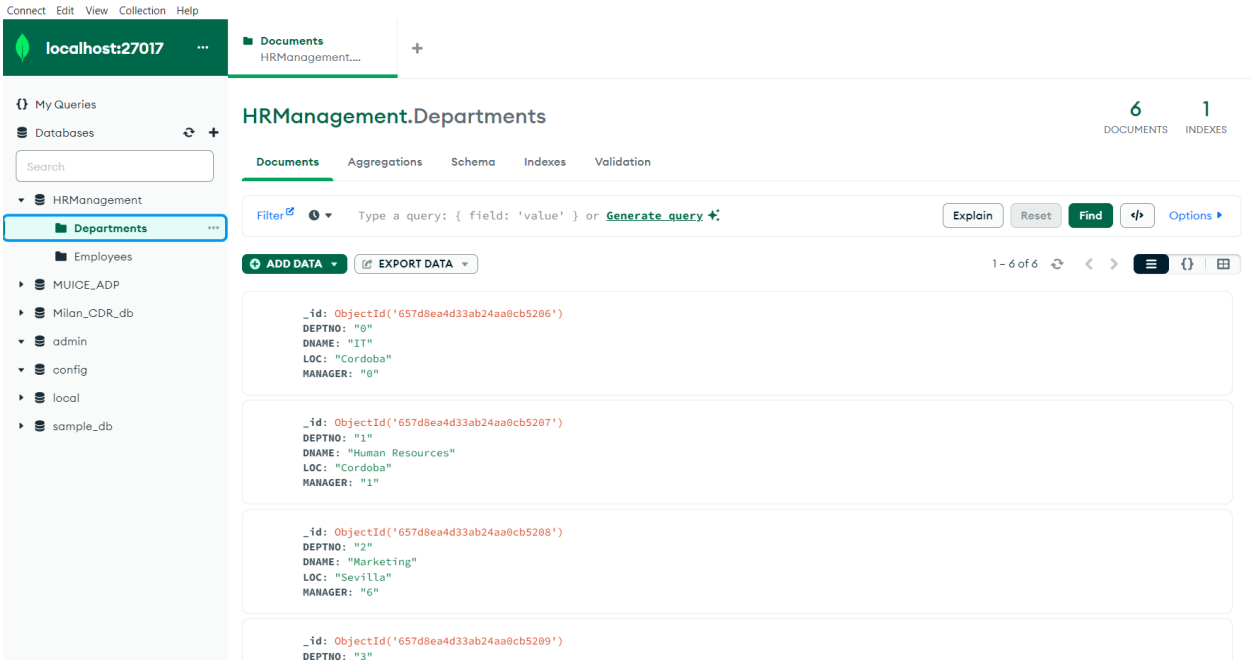
2.2. Carga de Información

Para cargar información en estas colecciones, se deben insertar documentos siguiendo la estructura propuesta anteriormente. El código utilizado para la carga de datos es el siguiente:

```
departments_data = [
    {"DEPTNO": "0", "DNAME": "IT", "LOC": "Cordoba", "MANAGER": "0"},
    {"DEPTNO": "1", "DNAME": "Human Resources", "LOC": "Cordoba", "MANAGER": "1"},
    {"DEPTNO": "2", "DNAME": "Marketing", "LOC": "Sevilla", "MANAGER": "6"},
    {"DEPTNO": "3", "DNAME": "Finance", "LOC": "Huelva", "MANAGER": "2"},
    {"DEPTNO": "4", "DNAME": "Operations", "LOC": "Cadiz", "MANAGER": "3"},
    {"DEPTNO": "5", "DNAME": "Customer Support", "LOC": "Valencia", "MANAGER": "4"},
]

employees_data = [
    {"EMPNO": "0", "ENAME": "Alba Marquez Rodriguez", "JOB": "Computing Engineer", "HIREDATE
    ": "2023-12-01",
    "SAL": 25000, "COMM": 1000, "DEPTNO": "0", "ISBOSS": True, "BOSS": None},
    {"EMPNO": "1", "ENAME": "Maria Isabel Hernandez Gonzalez", "JOB": "Human Resources", "
    HIREDATE": "2023-11-01",
    "SAL": 13000, "COMM": 1000, "DEPTNO": "1", "ISBOSS": True, "BOSS": None},
    {"EMPNO": "2", "ENAME": "Juana de la Cruz Santos", "JOB": "IT Technician", "HIREDATE": "
    2023-04-04",
    "SAL": 20000, "COMM": 1000, "DEPTNO": "0", "ISBOSS": False, "BOSS": 0},
    {"EMPNO": "3", "ENAME": "Jorge Perez Munoz", "JOB": "IT Technician", "HIREDATE": "
    2023-15-01",
    "SAL": 15000, "COMM": 1000, "DEPTNO": "1", "ISBOSS": False, "BOSS": 0},
    {"EMPNO": "4", "ENAME": "Lola Martinez Caraballo", "JOB": "IT Technician", "HIREDATE": "
    2023-14-02",
    "SAL": 13000, "COMM": 1000, "DEPTNO": "1", "ISBOSS": False, "BOSS": 0},
    {"EMPNO": "5", "ENAME": "Luis Moreno Diaz", "JOB": "Psicologist", "HIREDATE": "
    2023-18-02",
    "SAL": 12000, "COMM": 1000, "DEPTNO": "1", "ISBOSS": False, "BOSS": 1},
    {"EMPNO": "6", "ENAME": "Amanda Villa Lopez", "JOB": "Marketer", "HIREDATE": "2023-18-02
    ",
    "SAL": 23000, "COMM": 1000, "DEPTNO": "2", "ISBOSS": True, "BOSS": 0},
    {"EMPNO": "7", "ENAME": "Carlos Ruiz Gomez", "JOB": "Financial Analyst", "HIREDATE": "
    2023-05-10",
    "SAL": 18000, "COMM": 800, "DEPTNO": "3", "ISBOSS": True, "BOSS": None},
    {"EMPNO": "8", "ENAME": "Ana Jimenez Garcia", "JOB": "Operations Manager", "HIREDATE": "
    2023-06-15",
    "SAL": 22000, "COMM": 1200, "DEPTNO": "4", "ISBOSS": True, "BOSS": None},
    {"EMPNO": "9", "ENAME": "Miguel Serrano Navarro", "JOB": "Customer Support Specialist",
    "HIREDATE": "2023-04-20",
    "SAL": 16000, "COMM": 700, "DEPTNO": "5", "ISBOSS": True, "BOSS": None},
    {"EMPNO": "10", "ENAME": "Elena Rodriguez Fernandez", "JOB": "IT Analyst", "HIREDATE": "
    2023-05-01",
    "SAL": 19000, "COMM": 900, "DEPTNO": "0", "ISBOSS": False, "BOSS": 0},
    {"EMPNO": "11", "ENAME": "Pedro Sanchez Gomez", "JOB": "Marketing Specialist", "HIREDATE
    ": "2023-06-10",
    "SAL": 21000, "COMM": 1000, "DEPTNO": "2", "ISBOSS": False, "BOSS": 6},
    {"EMPNO": "12", "ENAME": "Sara Diaz Hernandez", "JOB": "Customer Support Representative"
    , "HIREDATE": "2023-07-01",
    "SAL": 17000, "COMM": 800, "DEPTNO": "5", "ISBOSS": False, "BOSS": 9},
    {"EMPNO": "13", "ENAME": "Antonio Lopez Ruiz", "JOB": "Finance Assistant", "HIREDATE": "
    2023-08-15",
    "SAL": 15000, "COMM": 700, "DEPTNO": "3", "ISBOSS": False, "BOSS": 7},
    {"EMPNO": "14", "ENAME": "Laura Martin Alonso", "JOB": "Operations Specialist", "
    HIREDATE": "2023-09-20",
    "SAL": 20000, "COMM": 1000, "DEPTNO": "4", "ISBOSS": False, "BOSS": 8},
    {"EMPNO": "15", "ENAME": "Pablo Garcia Rodriguez", "JOB": "IT Specialist", "HIREDATE": "
    2023-10-01",
```

```
    "SAL": 18000, "COMM": 900, "DEPTNO": "0", "ISBOSS": False, "BOSS": 10},
    {"EMPNO": "16", "ENAME": "Isabel Ramos Martinez", "JOB": "Customer Support
    Representative", "HIREDATE": "2023-11-10",
    "SAL": 16000, "COMM": 800, "DEPTNO": "5", "ISBOSS": False, "BOSS": 9},
    {"EMPNO": "17", "ENAME": "Manuel Torres Rodriguez", "JOB": "IT Analyst", "HIREDATE": "
    2023-12-01",
    "SAL": 18000, "COMM": 900, "DEPTNO": "0", "ISBOSS": False, "BOSS": 10},
    {"EMPNO": "18", "ENAME": "Silvia Fernandez Navarro", "JOB": "Marketing Coordinator", "
    HIREDATE": "2023-01-15",
    "SAL": 19000, "COMM": 1000, "DEPTNO": "2", "ISBOSS": False, "BOSS": 6},
    {"EMPNO": "19", "ENAME": "Rafael Lopez Diaz", "JOB": "Finance Analyst", "HIREDATE": "
    2023-02-20",
    "SAL": 17000, "COMM": 800, "DEPTNO": "3", "ISBOSS": False, "BOSS": 7},
    {"EMPNO": "20", "ENAME": "Carmen Serrano Ruiz", "JOB": "Operations Specialist", "
    HIREDATE": "2023-03-01",
    "SAL": 20000, "COMM": 1000, "DEPTNO": "4", "ISBOSS": False, "BOSS": 8},
    {"EMPNO": "21", "ENAME": "Diego Garcia Herrera", "JOB": "IT Specialist", "HIREDATE": "
    2023-04-10",
    "SAL": 19000, "COMM": 900, "DEPTNO": "0", "ISBOSS": False, "BOSS": 10},
    {"EMPNO": "22", "ENAME": "Natalia Diaz Martin", "JOB": "Customer Support Representative"
    , "HIREDATE": "2023-05-15",
    "SAL": 16000, "COMM": 800, "DEPTNO": "5", "ISBOSS": False, "BOSS": 9},
    {"EMPNO": "23", "ENAME": "Juan Perez Gomez", "JOB": "Marketing Specialist", "HIREDATE":
    "2023-06-01",
    "SAL": 21000, "COMM": 1000, "DEPTNO": "2", "ISBOSS": False, "BOSS": 6},
    {"EMPNO": "24", "ENAME": "Sofia Martin Torres", "JOB": "Finance Assistant", "HIREDATE":
    "2023-07-10",
    "SAL": 15000, "COMM": 700, "DEPTNO": "3", "ISBOSS": False, "BOSS": 7},
    {"EMPNO": "25", "ENAME": "Alejandro Ruiz Garcia", "JOB": "Operations Manager", "HIREDATE
    ": "2023-08-15",
    "SAL": 22000, "COMM": 1200, "DEPTNO": "4", "ISBOSS": False, "BOSS": 8},
]

# Insertar datos en la colecci n "Departments"
database['Departments'].insert_many(departments_data)

# Insertar datos en la colecci n "Employees"
database['Employees'].insert_many(employees_data)
```

Después podemos comprobar en MongoDB Compass si se han creado las colecciones y cargado los datos:



2.3. Consulta de datos

A continuación se presentan ejemplos de consultas que se han realizado para comprobar la implementación y creación de la base de datos.

2.3.1. Consulta para obtener todos los departamentos

Con esta consulta se obtienen todos los departamentos existentes, con sus atributos correspondientes.

```
all_departments = database['Departments'].find()
for department in all_departments:
    print(department)
```

```
{'_id': ObjectId('657d8ea4d33ab24aa0cb5206'), 'DEPTNO': '0', 'DNAME': 'IT', 'LOC': 'Cordoba'
    , 'MANAGER': '0'}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5207'), 'DEPTNO': '1', 'DNAME': 'Human Resources', '
    LOC': 'Cordoba', 'MANAGER': '1'}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5208'), 'DEPTNO': '2', 'DNAME': 'Marketing', 'LOC': '
    Sevilla', 'MANAGER': '6'}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5209'), 'DEPTNO': '3', 'DNAME': 'Finance', 'LOC': '
    Huelva', 'MANAGER': '2'}
{'_id': ObjectId('657d8ea4d33ab24aa0cb520a'), 'DEPTNO': '4', 'DNAME': 'Operations', 'LOC': '
    Cadiz', 'MANAGER': '3'}
{'_id': ObjectId('657d8ea4d33ab24aa0cb520b'), 'DEPTNO': '5', 'DNAME': 'Customer Support', '
    LOC': 'Valencia', 'MANAGER': '4'}
```

2.3.2. Consulta para obtener todos los empleados

Con esta consulta se obtienen todos los empleados de la base de datos.

```
all_employees = database['Employees'].find()
for employee in all_employees:
    print(employee)
```

```
{'_id': ObjectId('657d8ea4d33ab24aa0cb520c'), 'EMPNO': '0', 'ENAME': 'Alba Marquez Rodriguez
    ', 'JOB': 'Computing Engineer', 'HIREDATE': '2023-12-01', 'SAL': 25000, 'COMM': 1000, '
    DEPTNO': '0', 'ISBOSS': True, 'BOSS': None}
{'_id': ObjectId('657d8ea4d33ab24aa0cb520d'), 'EMPNO': '1', 'ENAME': 'Maria Isabel Hernandez
     Gonzalez', 'JOB': 'Human Resources', 'HIREDATE': '2023-11-01', 'SAL': 13000, 'COMM':
    1000, 'DEPTNO': '1', 'ISBOSS': True, 'BOSS': None}
...
```

2.3.3. Consulta para obtener los empleados de un departamento específico (ejemplo con DEPTNO = '0')

Si se quieren obtener los empleados de un departamento en concreto esta consulta puede usarse cambiando el número de departamento.

```
it_department_employees = database['Employees'].find({"DEPTNO": "0"})
for employee in it_department_employees:
    print(employee)
```

```
{'_id': ObjectId('657d8ea4d33ab24aa0cb520c'), 'EMPNO': '0', 'ENAME': 'Alba Marquez Rodriguez
    ', 'JOB': 'Computing Engineer', 'HIREDATE': '2023-12-01', 'SAL': 25000, 'COMM': 1000, '
    DEPTNO': '0', 'ISBOSS': True, 'BOSS': None}
{'_id': ObjectId('657d8ea4d33ab24aa0cb520e'), 'EMPNO': '2', 'ENAME': 'Juana de la Cruz
    Santos', 'JOB': 'IT Technician', 'HIREDATE': '2023-04-04', 'SAL': 20000, 'COMM': 1000, '
    DEPTNO': '0', 'ISBOSS': False, 'BOSS': 0}
...
```

2.3.4. Consulta para obtener el nombre del jefe de un departamento (ejemplo con DEPTNO = '2')

Esta consulta permite obtener únicamente el nombre del jefe de un departamento y puede modificarse el número de departamento para obtener el de cualquier departamento.

```
marketing_department_boss = database['Employees'].find_one({"DEPTNO": "2", "ISBOSS": True})
print(marketing_department_boss['ENAME'])
```

```
Amanda Villa Lopez
```

2.3.5. Consulta para obtener la cantidad total de empleados

Esta consulta cuenta el número de empleados totales.

```
total_employees_count = database['Employees'].count_documents({})
print(f"Total de empleados: {total_employees_count}")
```

```
Total de empleados: 26
```

2.3.6. Consulta para obtener la cantidad total de empleados de un departamento (ejemplo con 'DEPTNO' = 0

Esta consulta cuenta el número de empleados totales de un departamento en específico. En concreto esta cosulta obtiene el número de empleados del departamento de IT.

```
it_department_employees_count = database['Employees'].count_documents({"DEPTNO": "0"})
print(f"Total de empleados del departamento de IT: {it_department_employees_count}")
```

```
Total de empleados del departamento de IT: 6
```

2.3.7. Consulta para obtener los empleados cuyo salario sea mayor a 20000

Esta consulta filtra entre todos los empleados aquellos cuyo salario supera una cifra.

```
high_salary_employees = database['Employees'].find({"SAL": {"$gt": 20000}})
for employee in high_salary_employees:
    print(employee)
```

```
{'_id': ObjectId('657d8ea4d33ab24aa0cb520c'), 'EMPNO': '0', 'ENAME': 'Alba Marquez Rodriguez
    ', 'JOB': 'Computing Engineer', 'HIREDATE': '2023-12-01', 'SAL': 25000, 'COMM': 1000, '
    DEPTNO': '0', 'ISBOSS': True, 'BOSS': None}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5212'), 'EMPNO': '6', 'ENAME': 'Amanda Villa Lopez', '
    JOB': 'Marketer', 'HIREDATE': '2023-18-02', 'SAL': 23000, 'COMM': 1000, 'DEPTNO': '2', '
    ISBOSS': True, 'BOSS': 0}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5214'), 'EMPNO': '8', 'ENAME': 'Ana Jimenez Garcia', '
    JOB': 'Operations Manager', 'HIREDATE': '2023-06-15', 'SAL': 22000, 'COMM': 1200, '
    DEPTNO': '4', 'ISBOSS': True, 'BOSS': None}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5217'), 'EMPNO': '11', 'ENAME': 'Pedro Sanchez Gomez',
     'JOB': 'Marketing Specialist', 'HIREDATE': '2023-06-10', 'SAL': 21000, 'COMM': 1000, '
    DEPTNO': '2', 'ISBOSS': False, 'BOSS': 6}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5223'), 'EMPNO': '23', 'ENAME': 'Juan Perez Gomez', '
    JOB': 'Marketing Specialist', 'HIREDATE': '2023-06-01', 'SAL': 21000, 'COMM': 1000, '
    DEPTNO': '2', 'ISBOSS': False, 'BOSS': 6}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5225'), 'EMPNO': '25', 'ENAME': 'Alejandro Ruiz Garcia
    ', 'JOB': 'Operations Manager', 'HIREDATE': '2023-08-15', 'SAL': 22000, 'COMM': 1200, '
    DEPTNO': '4', 'ISBOSS': False, 'BOSS': 8}
```

2.3.8. Consulta para obtener los departamentos en una ubicación específica (ejemplo con LOC = 'Cordoba')

Consulta para mostrar los departamentos que se encuentren en una localización específica.

```
cordoba_departments = database['Departments'].find({"LOC": "Cordoba"})
for department in cordoba_departments:
    print(department)
```

```
{'_id': ObjectId('657d8ea4d33ab24aa0cb5206'), 'DEPTNO': '0', 'DNAME': 'IT', 'LOC': 'Cordoba'
    , 'MANAGER': '0'}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5207'), 'DEPTNO': '1', 'DNAME': 'Human Resources', '
    LOC': 'Cordoba', 'MANAGER': '1'}
```

2.3.9. Consulta para obtener los empleados que fueron contratados después de cierta fecha (ejemplo con HIREDATE después de '2023-06-01')

```
recently_hired_employees = database['Employees'].find({"HIREDATE": {"$gt": "2023-06-01"}})
for employee in recently_hired_employees:
    print(employee)
```

```
{'_id': ObjectId('657d8ea4d33ab24aa0cb520c'), 'EMPNO': '0', 'ENAME': 'Alba Marquez Rodriguez
    ', 'JOB': 'Computing Engineer', 'HIREDATE': '2023-12-01', 'SAL': 25000, 'COMM': 1000, '
    DEPTNO': '0', 'ISBOSS': True, 'BOSS': None}
{'_id': ObjectId('657d8ea4d33ab24aa0cb520d'), 'EMPNO': '1', 'ENAME': 'Maria Isabel Hernandez
     Gonzalez', 'JOB': 'Human Resources', 'HIREDATE': '2023-11-01', 'SAL': 13000, 'COMM':
    1000, 'DEPTNO': '1', 'ISBOSS': True, 'BOSS': None}
{'_id': ObjectId('657d8ea4d33ab24aa0cb520f'), 'EMPNO': '3', 'ENAME': 'Jorge Perez Munoz', '
    JOB': 'IT Technician', 'HIREDATE': '2023-15-01', 'SAL': 15000, 'COMM': 1000, 'DEPTNO': '
    1', 'ISBOSS': False, 'BOSS': 0}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5210'), 'EMPNO': '4', 'ENAME': 'Lola Martinez
    Caraballo', 'JOB': 'IT Technician', 'HIREDATE': '2023-14-02', 'SAL': 13000, 'COMM':
    1000, 'DEPTNO': '1', 'ISBOSS': False, 'BOSS': 0}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5211'), 'EMPNO': '5', 'ENAME': 'Luis Moreno Diaz', '
    JOB': 'Psicologist', 'HIREDATE': '2023-18-02', 'SAL': 12000, 'COMM': 1000, 'DEPTNO': '1'
    , 'ISBOSS': False, 'BOSS': 1}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5212'), 'EMPNO': '6', 'ENAME': 'Amanda Villa Lopez', '
    JOB': 'Marketer', 'HIREDATE': '2023-18-02', 'SAL': 23000, 'COMM': 1000, 'DEPTNO': '2', '
    ISBOSS': True, 'BOSS': 0}
...
```

2.3.10. Consulta para obtener los empleados que son jefes (ISBOSS=True)

```
boss_employees = database['Employees'].find({"ISBOSS": True})
for boss in boss_employees:
    print(boss)
```

```
{'_id': ObjectId('657d8ea4d33ab24aa0cb520c'), 'EMPNO': '0', 'ENAME': 'Alba Marquez Rodriguez
    ', 'JOB': 'Computing Engineer', 'HIREDATE': '2023-12-01', 'SAL': 25000, 'COMM': 1000, '
    DEPTNO': '0', 'ISBOSS': True, 'BOSS': None}
{'_id': ObjectId('657d8ea4d33ab24aa0cb520d'), 'EMPNO': '1', 'ENAME': 'Maria Isabel Hernandez
     Gonzalez', 'JOB': 'Human Resources', 'HIREDATE': '2023-11-01', 'SAL': 13000, 'COMM':
    1000, 'DEPTNO': '1', 'ISBOSS': True, 'BOSS': None}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5212'), 'EMPNO': '6', 'ENAME': 'Amanda Villa Lopez', '
    JOB': 'Marketer', 'HIREDATE': '2023-18-02', 'SAL': 23000, 'COMM': 1000, 'DEPTNO': '2', '
    ISBOSS': True, 'BOSS': 0}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5213'), 'EMPNO': '7', 'ENAME': 'Carlos Ruiz Gomez', '
    JOB': 'Financial Analyst', 'HIREDATE': '2023-05-10', 'SAL': 18000, 'COMM': 800, 'DEPTNO'
    : '3', 'ISBOSS': True, 'BOSS': None}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5214'), 'EMPNO': '8', 'ENAME': 'Ana Jimenez Garcia', '
    JOB': 'Operations Manager', 'HIREDATE': '2023-06-15', 'SAL': 22000, 'COMM': 1200, '
    DEPTNO': '4', 'ISBOSS': True, 'BOSS': None}
{'_id': ObjectId('657d8ea4d33ab24aa0cb5215'), 'EMPNO': '9', 'ENAME': 'Miguel Serrano Navarro
    ', 'JOB': 'Customer Support Specialist', 'HIREDATE': '2023-04-20', 'SAL': 16000, 'COMM':
     700, 'DEPTNO': '5', 'ISBOSS': True, 'BOSS': None}
```

3. Fase 2: Aplicaciones científicas y empresariales

Utilizando las bases de datos de compuestos del ejemplo práctico desarrollado en la parte teórica de la asignatura, se ha implementado el código python necesario para predecir (clasificación) el tipo de actividad biológica (Active/Inactive).

3.1. Carga de datos

La carga de datos en el servidor MongoDB se ha realizado siguiendo las indicaciones, a partir del fichero descargado `dump` y el comando de mongo para restaurar la base de datos.

3.2. Preprocesamiento de datos

Luego se ha llevado a cabo un preprocesamiento de datos para poder tenerlos en un formato con el que poder realizar una clasificación. Se han seguido algunas de las indicaciones propuestas en el enunciado de la actividad.

1. Cargar los datos desde la base de datos MongoDB

```
connection_details = MONGO_LINK

# Databases
db_CDS29_name='CDS29';
db_CDS16_name='CDS16';

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

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

def load_database(db_name):
    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()

    return db

db_CDS29=load_database(db_CDS29_name);
db_CDS16=load_database(db_CDS16_name);

```

2. Cargar los datos desde las colecciones necesarias (molecules)

```
collection_name = 'molecules'

# Funci n para cargar datos desde una colecci n espec fica
def load_data(db, collection_name):
    collection = db[collection_name]
    cursor = collection.find()
    data = pd.DataFrame(list(cursor))
    return data

# Cargar datos de las colecciones molecules
data_CDS29 = load_data(db_CDS29, collection_name)
data_CDS16 = load_data(db_CDS16, collection_name)

```

Si se visualizan los datos cargados se obtiene lo siguiente:

_id	smiles	class	MOLECULEID	rdmol	mfp
8.0	CNC(=O)c1cc(Oc2ccc(NC(=O)Nc3ccc(Cl)c(C(F)(F)F)...	Inactive	M110015	b'000000000b0000...	'bits':[9, 33, 53, 56, 114, 128, 153, 184, 1...
3.0	O=C(O)CC(O)CC(O)C=Cc1c(C2CC2)nc2ccccc2c1-c1ccc...	Inactive	M3412451	b'000000000b0000...	'bits': [1, 25, 64, 73, 80, 90, 105, 117, 136...

Cuadro 1: Ejemplo de datos pertenecientes a CDS16

3. Convertir las posiciones de los bits en fingerprints moleculares completos.

```
collection_name = 'molecules'

# Preprocesamiento de datos
def expand_fingerprint(mfp_data, size=1024):
    bits = mfp_data.get('bits', [])  # Acceder al diccionario bits en mfp
    fingerprint = np.zeros(size)
    fingerprint[bits] = 1
    return fingerprint

# Aplicar la funci n a cada fila
data_CDS29['fingerprint'] = data_CDS29['mfp'].apply(lambda mfp_data: expand_fingerprint
    (mfp_data))
data_CDS16['fingerprint'] = data_CDS16['mfp'].apply(lambda mfp_data: expand_fingerprint
    (mfp_data))

```

4. Separar los datos en variables predictoras (`X`) y variable respuesta (`y`).

```
# Variables predictoras y variable respuesta
X_CDS29 = pd.DataFrame(data_CDS29['fingerprint'].tolist())
X_CDS16 = pd.DataFrame(data_CDS16['fingerprint'].tolist())

# No se han eliminado variables predictoras (las que tienen los valores a 0)
y_CDS29 = data_CDS29['class'].map({'Active': 1, 'Inactive': 0})
y_CDS16 = data_CDS16['class'].map({'Active': 1, 'Inactive': 0})

```

En este apartado se recomendaba eliminar las columnas en las que todos los valores eran '0'. Sin embargo, se han decidido mantener todas las columnas.

Tras realizar la separación en variables predictoras y respuestas se obtiene un array similar al que el enunciado adelantaba, esta sería la parte correspondiente a las variables predictoras de una de las bases de datos:

	0	1	2	3	4	5	6	7	8	9	...	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figura 1: Parte de variables predictoras de CDS29

Y esta la parte correspondiente a las variables respuesta, donde 0 corresponde con Inactiva y 1 con Activa:

0	0
1	0
2	0
3	0
4	0

Figura 2: Parte de variables respuesta de CDS29

3.3. Preparación del modelo clasificador

Para realizar un modelo clasificador es necesario disponer de un conjunto de datos y dividirlo en datos de entrenamiento y de prueba. Además, se usará la librería *scikit-learn* para realizar modelos clasificadores.

Para eso es necesario también tenerlo dividido entre X e Y. Es decir, entre variables predictoras y respuesta, lo cuál ya se ha realizado en la parte del preprocesamiento.

Antes de realizar ningún tipo de separación de los subconjuntos se va a analizar brevemente la distribución de las clases:

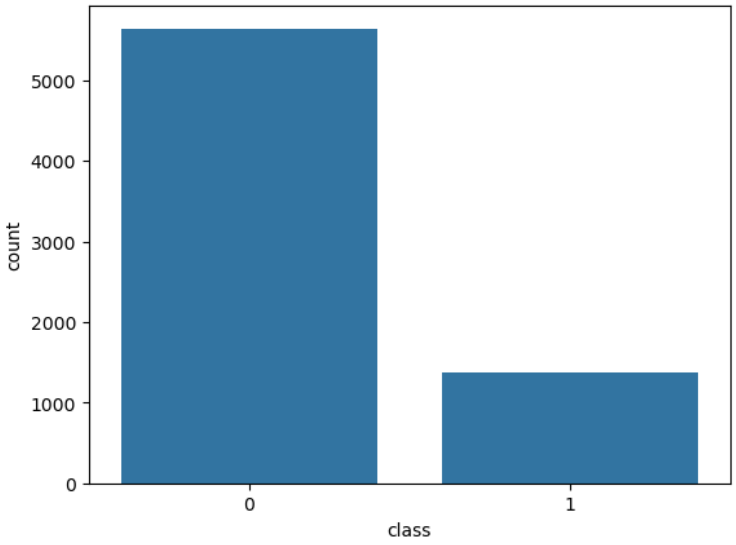


Figura 3: Distribución de clases de CDS29

Como se puede ver en el conjunto de datos `CDS29` están desbalanceadas por lo que la división de los datos se realizará con estratificación. Sin embargo, el desbalanceo es muy alto y puede conllevar a una muy mala clasificación de aquellas clases pertenecientes a la clase minoritaria, en este caso, la clase `Active: 1`.

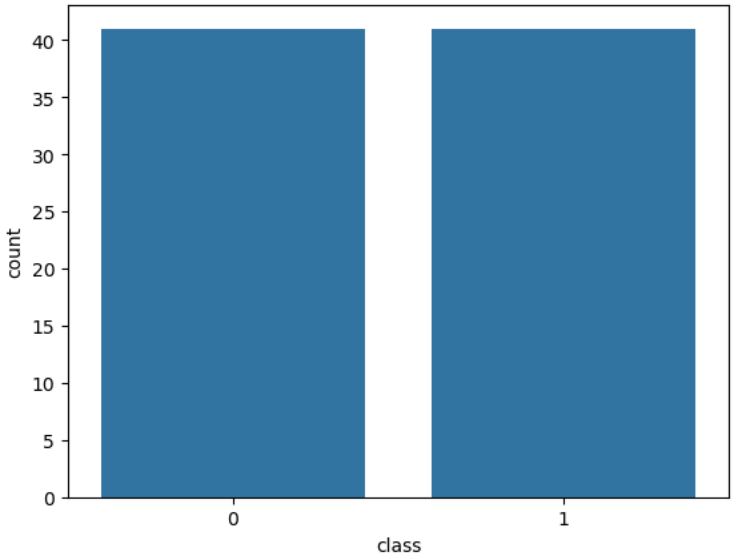


Figura 4: Distribución de clases de CDS16

Por otro lado, en el conjunto de datos `CDS16` las clases están más equilibradas, sin embargo, el problema es la poca cantidad de clases que tiene este conjunto de datos. Esto podría reflejarse en el rendimiento de los modelos de clasificación al no tener suficientes datos para conseguir un buen modelo entrenado.

1. Dividir los datos en conjuntos de entrenamiento y prueba.

   Utilizando el siguiente código se han dividido ambos conjuntos de datos.

```
# Dividir en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
    =42,stratify=y)

```

La distribución ha sido la siguiente, en azul queda reflejado el conjunto de entrenamiento y en rojo el de test:

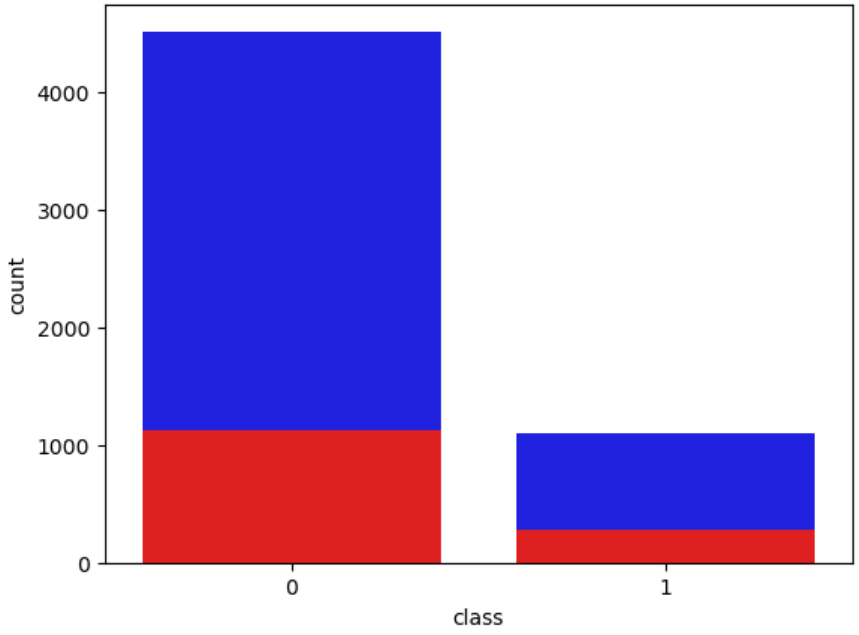


Figura 5: Distribución subconjuntos CDS29

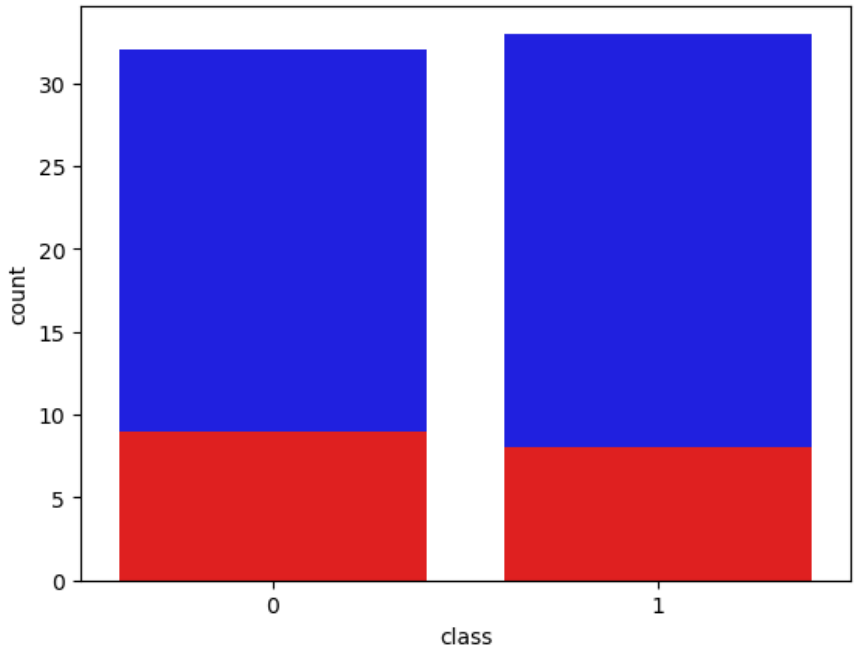


Figura 6: Distribución subconjuntos CDS16

2. Elegir un modelo de clasificación de scikit-learn.

   Se va a realizar una comparativa de diferentes algoritmos, se han elegido 3 modelos y de uno de ellos se ha hecho una modificación de los hiperparámetros.

   Los modelos elegidos son:

   - Random Forest (predetermiado y 100 árboles con 10 de profudidad)
   - Naive Bayes
   - SVC

3. Entrenar el modelo con los datos de entrenamiento.

```
# 1. Random Forest
model_rf = RandomForestClassifier(random_state=42)
model_rf.fit(X_train, y_train)

# 2. Random Forest con 100  rboles  y profundidad m xima de 10
model_rf_100 = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
model_rf_100.fit(X_train, y_train)

# 3. Naive Bayes
model_nb = GaussianNB()
model_nb.fit(X_train, y_train)

# 4. SVC
model_svm = SVC(random_state=42)
model_svm.fit(X_train, y_train)

```

3.4. Evaluación del modelo:

1. Realizar predicciones en el conjunto de prueba

```
# Predicciones
y_pred_rf = model_rf.predict(X_test)
y_pred_rf_100 = model_rf_100.predict(X_test)
y_pred_nb = model_nb.predict(X_test)
y_pred_svc = model_svc.predict(X_test)

```

2. Calcular métricas

```
# Reporte de clasificaci n
print('Classification Report Random Forest: \n', classification_report(y_test,
    y_pred_rf))
print('\n\n\n\n\n\nClassification Report Random Forest 100: \n', classification_report(
    y_test, y_pred_rf_100))
print('\n\n\n\n\n\nClassification Report Naive Bayes: \n', classification_report(y_test
    , y_pred_nb))
print('\n\n\n\n\n\nClassification Report SVC: \n', classification_report(y_test,
    y_pred_svc))

# Random Forest
cm_rf = confusion_matrix(y_test, y_pred_rf)
print('Confusion Matrix Random Forest: \n', cm_rf)
cm_display = ConfusionMatrixDisplay(cm_rf).plot()
plt.show()
# Matriz Normalizada
cm_rf_norm = cm_rf.astype('float') / cm_rf.sum(axis=1)[:, np.newaxis]
print('Confusion Matrix Normalizada Random Forest: \n', cm_rf_norm)
cm_display = ConfusionMatrixDisplay(cm_rf_norm).plot()
plt.show()

print("\n\n\n==================================================\n\n\n")

```

```
# Random Forest con 100  rboles  y profundidad m xima de 10
cm_rf_100 = confusion_matrix(y_test, y_pred_rf_100)
print('Confusion Matrix Random Forest 100: \n', cm_rf_100)
cm_display = ConfusionMatrixDisplay(cm_rf_100).plot()
plt.show()
# Matriz Normalizada
cm_rf_100_norm = cm_rf_100.astype('float') / cm_rf_100.sum(axis=1)[:, np.newaxis]
print('Confusion Matrix Normalizada Random Forest 100: \n', cm_rf_100_norm)
cm_display = ConfusionMatrixDisplay(cm_rf_100_norm).plot()
plt.show()

print("\n\n\n==================================================\n\n\n")

# Naive Bayes
cm_nb = confusion_matrix(y_test, y_pred_nb)
print('Confusion Matrix Naive Bayes: \n', cm_nb)
cm_display = ConfusionMatrixDisplay(cm_nb).plot()
plt.show()
# Matriz Normalizada
cm_nb_norm = cm_nb.astype('float') / cm_nb.sum(axis=1)[:, np.newaxis]
print('Confusion Matrix Normalizada Naive Bayes: \n', cm_nb_norm)
cm_display = ConfusionMatrixDisplay(cm_nb_norm).plot()
plt.show()

print("\n\n\n==================================================\n\n\n")

# SVC
cm_svc = confusion_matrix(y_test, y_pred_svc)
print('Confusion Matrix SVC: \n', cm_svc)
cm_display = ConfusionMatrixDisplay(cm_svc).plot()
plt.show()
# Matriz Normalizada
cm_svc_norm = cm_svc.astype('float') / cm_svc.sum(axis=1)[:, np.newaxis]
print('Confusion Matrix Normalizada SVC: \n', cm_svc_norm)
cm_display = ConfusionMatrixDisplay(cm_svc_norm).plot()
plt.show()

```

3.5. Análisis de los resultados

Para comenzar el análisis se va a empezar analizando el reporte de clasificación que muestra varias métricas importantes para comprender el funcionamiento de cada algoritmo y modelo entrenado. Además se acompañará de las matrices de confusión.

Primero se analizarán los resultados del conjunto de datos `CDS29` y posteriormente del otro conjunto de datos `CDS16`.

3.5.1. Conjunto de Datos `CDS29`

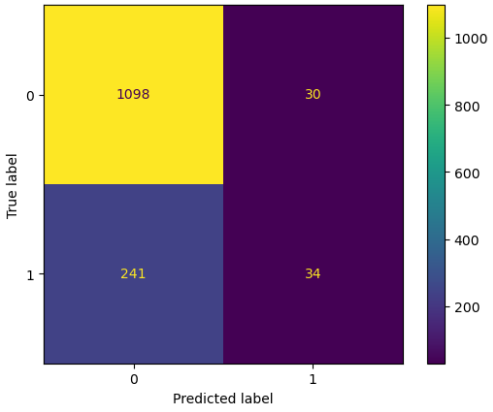
El primer algoritmo es Random Forest, con los hiperparámetros por defecto. Tras entrenar el modelo y realizar predicciones, las métricas obtenidas en el *clasification report* que *scikit-learn* facilita son las siguientes:

Class	Precision	Recall	F1-Score	Support
0	0.82	0.97	0.89	1128
1	0.53	0.12	0.20	275
Accuracy		0.81 (1403)		
Macro Avg		0.55 (0.55)	(0.55)	(1403)
Weighted Avg		0.81 (0.81)	(0.75)	(1403)

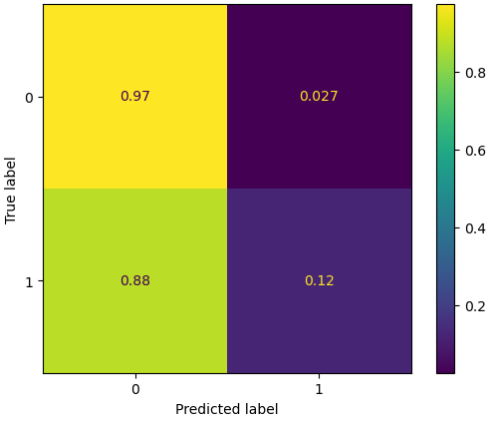
Cuadro 2: Classification Report Random Forest

Se pueden ver de manera más gráfica en las matrices de confusión. Que muestra la distribución de Falsos Positivos, Falsos Negativos, Verdaderos Positivos y Verdaderos Negativos que se obtienen en la predicción. Al tener el conjunto de datos desbalanceado es interesante normalizar la matriz de confusión. Así el mapeo

de los colores es más fácil de interpretar, ya que se tienen en cuenta respecto al número de instancias de cada clase y no del total.



Cuadro 3: Matriz de Confusión



Cuadro 4: Matriz de Confusión Normalizada

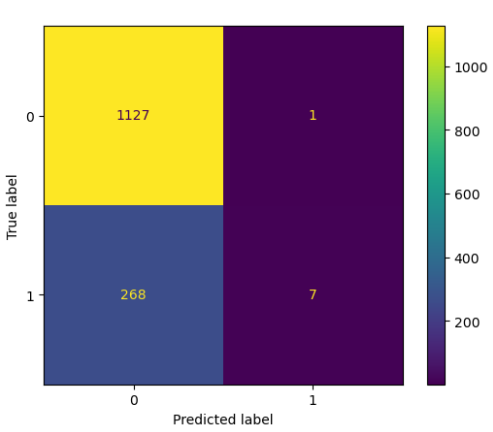
Cuadro 5: Matrices de Confusión de Random Forest

Para la clase mayoritaria (0), se observan métricas con valoresmuy positivos, con una precisión del 82 %, un recall del 97 % y un F1-score del 89 %, indicando una gran capacidad del modelo para identificar instancias de esta clase. Sin embargo, para la clase minoritaria (1), la precisión, aunque aceptable en 53 %, se ve acompañada de valores muy bajos en recall (12 %) y F1-score (20 %), que muestran dificultades significativas en la clasificación de esta clase. A pesar de un accuracy global del 81 %, el desbalanceo de instancias entre las clases se refleja en las métricas de Macro Average y Weighted Average, indicando la necesidad de mejorar la clasificación de la clase minoritaria. Es muy importante abordar este problema, ya que simplemente cambiar algoritmos y ajustar parámetros puede no ser suficiente sin un dataset balanceado.

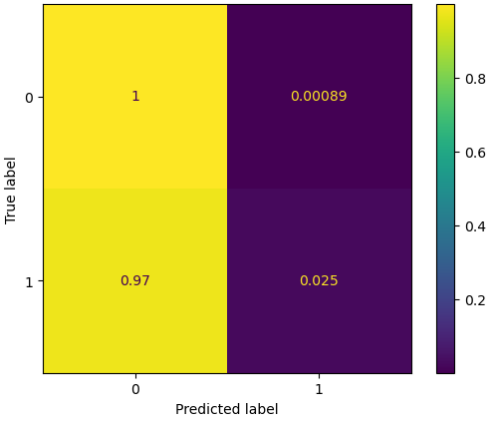
A continuación se cambian algunos parámetros del algoritmo Random Forest para intentar explorar la posibilidad de mejora del rendimiento a través del cambio de algunos parámetros:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.81	1.00	0.89	1128
1	0.88	0.03	0.05	275
Accuracy		0.81 (1403)		
Macro Avg		0.84 (0.51)	(0.47)	(1403)
Weighted Avg		0.82 (0.81)	(0.73)	(1403)

Cuadro 6: Classification Report Random Forest 100



Cuadro 7: Matriz de Confusión



Cuadro 8: Matriz de Confusión Normalizada

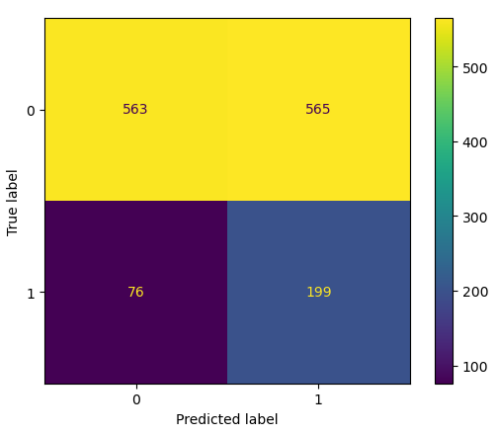
Cuadro 9: Matrices de Confusión de Random Forest 100

Para la clase mayoritaria (0), las métricas muestran resultados muy buenos, con una precisión del 81 %, un recall del 100 %, y un F1-score del 89 %, resaltando la eficacia del modelo en la identificación de instancias de esta clase. Es especialmente llamativo el valor del recall para esta clase, que quiere decir que no ha habido falsos negativos. Sin embargo, para la clase minoritaria (1), la precisión, aunque razonable en 88 %, se ve comprometida por valores muy bajos en recall (3 %) y F1-score (5 %), indicando graves problemas en la clasificación de esta categoría. Una precisión del 88 % significa que el 88 % de las instancias clasificadas como pertenecientes a esa clase son correctas. Sin embargo, un recall del 3 % indica que solo el 3 % de todas las instancias reales de esa clase fueron identificadas correctamente. Esto quiere decir que muy pocas instancias han sido clasificadas como la clase 1, pero que aquellas clasificadas como esta, han sido clasificadas correctamente. Por otro lado, a pesar de un accuracy global del 81 %, la disparidad en el número de instancias entre las clases se refleja en las métricas de Macro Average y Weighted Average, subrayando la necesidad de mejorar la clasificación de la clase minoritaria, que no ha podido ser solucionado cambiando los parámetros. Es importante saber cuál es el objetivo final, pues si lo importante es que aquellas clasificaciones de la clase 1 sean correctas, este experimento habría conseguido su objetivo.

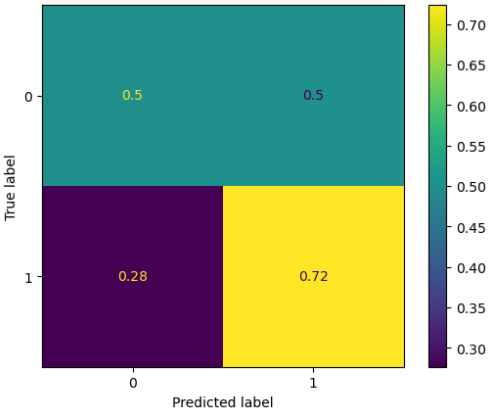
A continuación el *clasification report* del algoritmo Naive Bayes:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.88	0.50	0.64	1128
1	0.26	0.72	0.38	275
Accuracy		0.54 (1403)		
Macro Avg		0.57 (0.61)	(0.51)	(1403)
Weighted Avg		0.76 (0.54)	(0.59)	(1403)

Cuadro 10: Classification Report Naive Bayes



Cuadro 11: Matriz de Confusión



Cuadro 12: Matriz de Confusión Normalizada

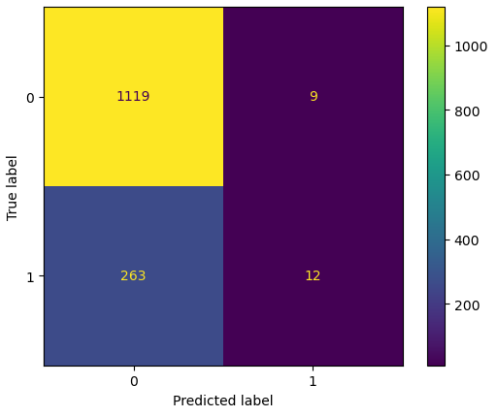
Cuadro 13: Matrices de Confusión de Naive Bayes

Para la clase mayoritaria (0), las métricas muestran resultados buenos, con una precisión del 88 %, un recall del 50 %, y un F1-score del 64 %, que muestra un problema similar al encontrado para la clase 1 con el algoritmo anterior. Pocas instancias son clasificadas como la clase 0. Pero aquellas clasificadas como la 0, son realmente esta clase. Sin embargo, para la clase minoritaria (1), la precisión es del 26 %, el del recall (72 %) y F1-score (38 %), esta vez hay más instancias clasificadas como la clase 1, pero sin embargo, pocas son realmente pertenecientes a esta clase. Por otro lado, el accuracy global esta vez es del 54 %. Mostrando aún problemas para una buena clasificación de las clases.

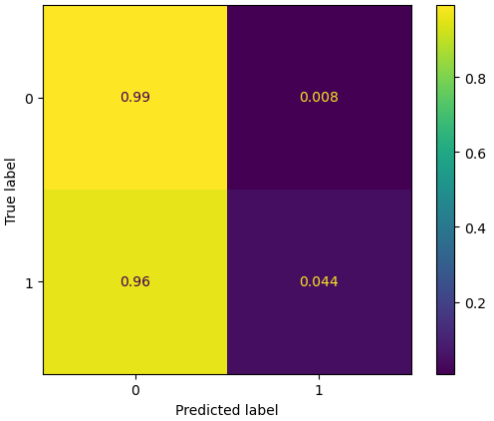
A continuación el *clasification report* del algoritmo SVC:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.81	0.99	0.89	1128
1	0.57	0.04	0.08	275
Accuracy		0.81 (1403)		
Macro Avg		0.69 (0.52)	(0.49)	(1403)
Weighted Avg		0.76 (0.81)	(0.73)	(1403)

Cuadro 14: Classification Report SVC



Cuadro 15: Matriz de Confusión



Cuadro 16: Matriz Normalizada

Cuadro 17: Matrices de Confusión de SVC

Para la clase mayoritaria (0), las métricas muestran resultados muy buenos, con una precisión del 81 %, un recall del 99 %, y un F1-score del 89 %. Sin embargo, para la clase minoritaria (1), la precisión es del 57 %, el del recall (4 %) y F1-score (8 %), mostrando el mismo problema que se experimentaba con el algoritmo de Random Forest. El accuracy global esta vez es del 81 %. Mostrando aún problemas para una buena clasificación de las clases.

En términos globales se podría decir que el algoritmo con mejores resultados serían el Random Forest y SVC. Sin embargo, dependería del problema y qué métrica interesa más. Por ejemplo, en el caso de querer identificar con total seguridad aquellas clases positivas (1) el Random Forest con los parámetros ajustados sería la mejor opción.

3.5.2. Conjunto de Datos `CDS16`

El conjunto de datos anterior tenía el problema del desbalanceo de clases. Por ello se ha conseguido muy buenos resultados en la clase con mayor número de instancias. Los resultados sobre el conjunto de datos que se va a analizar a continuación tienen otro problema. Las clases estaban balanceadas, sin embargo, el número de instancias es muy bajo tal y como se pudo ver el los gráficos anteriormente mostrados.

En esta ocasión se analizará del mismo modo las métricas obtenidas co el *classification report* que facilita *scikit-learn*. Sin embargo, al tener un conjunto de datos balanceado, esta vez no merece la pena realizar e incluír la normalización de las matrices de confusiones. Por lo que se añadirá directamente la matriz de confusión de cada algoritmo sin normalizar.

El primer algoritmo es Random Forest, con los hiperparámetros por defecto. Tras entrenar el modelo y realizar predicciones, las métricas obtenidas en el *clasification report* que *scikit-learn* facilita son las siguientes:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.89	0.89	0.89	9
1	0.88	0.88	0.88	8
Accuracy		0.88 (17)		
Macro Avg		0.88 (0.88) (0.88) (17)		
Weighted Avg		0.88 (0.88) (0.88) (17)		

Cuadro 18: Classification Report Random Forest

Se pueden ver de manera más gráfica en la matriz de confusión. Que muestra la distribución de Falsos Positivos, Falsos Negativos, Verdaderos Positivos y Verdaderos Negativos que se obtienen en la predicción.

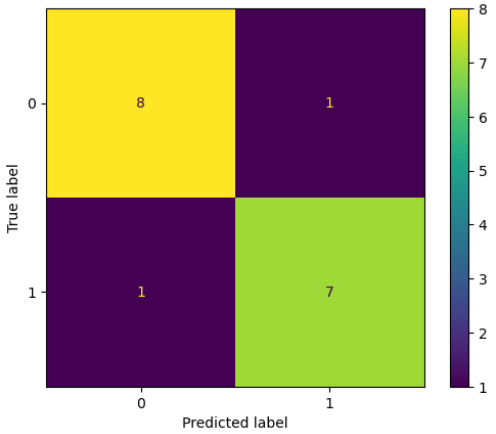


Figura 7: Matriz Confusión Random Forest

Esta vez con observar rápidamente la matriz de confusión que es mucho más visual ya se puede saber que los resultados han sido muy positivos, pues se puede ver como se consigue la diagonal principal deseada en una matriz de confusión. Entrando a analizar en más detalle los valores de las métricas, ambas clases estaban balanceadas aunque con muy poca cantidad de datos, se observan métricas con valores muy positivos para ambas clases, con una precisión, un recall, un F1-score del 89 % en la clase (0). En el caso de la clase (1) el valor de estas métricas es casi el mismo, del 88 %. Lo que quiere decir que el modelo está muy equilibrado y balanceado en cuanto al rendimiento entre ambas clases. Por ello las métricas globales como el *accuracy* también son muy positivas, del 88 % en todos los casos también. Este modelo es muy equilibrado y ha conseguido muy buen rendimiento en ambas clases.

A continuación se cambian algunos parámetros del algoritmo Random Forest para intentar explorar la posibilidad de mejora o empeoramiento del rendimiento a través del cambio de algunos parámetros:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.89	0.89	0.89	9
1	0.88	0.88	0.88	8
Accuracy		0.88 (17)		
Macro Avg		0.88 (0.88) (0.88) (17)		
Weighted Avg		0.88 (0.88) (0.88) (17)		

Cuadro 19: Classification Report Random Forest 100

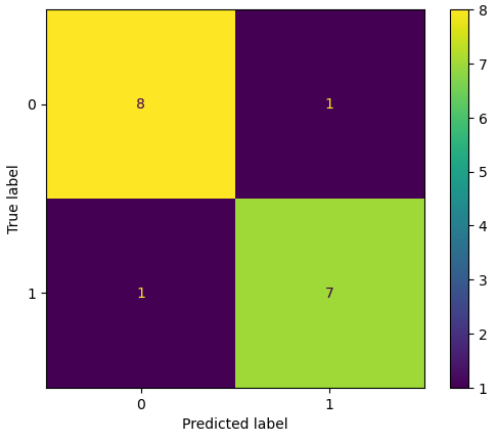


Figura 8: Matriz Confusión Random Forest 100

En este caso los resultados, al menos en cuanto al número de instancias clasificadas correcta o incorrectamente, son idénticos a las del algoritmo Random Forest anterior. Lo que puede indicar que teniendo un número muy pequeño de instancias quizás no merece la pena explorar en profundidad la posivilidad de cambiar hiperparámetros. Al menos, no en este algoritmo.

A continuación el *clasification report* del algoritmo Naive Bayes:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.80	0.89	0.84	9
1	0.86	0.75	0.80	8
Accuracy		0.82 (17)		
Macro Avg		0.83 (0.82) (0.82) (17)		
Weighted Avg		0.83 (0.82) (0.82) (17)		

Cuadro 20: Classification Report Naive Bayes

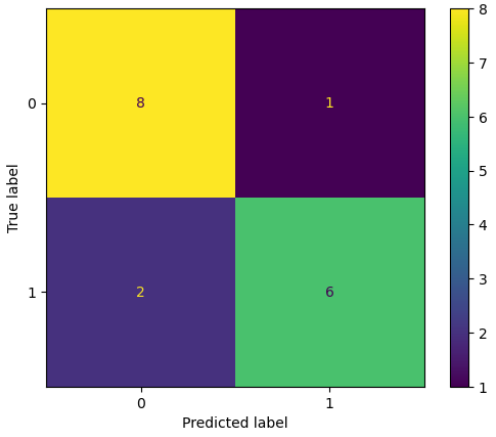


Figura 9: Matriz Confusión Naive Bayes

En este caso los resultados vuelven a ser muy buenos aunque no superan en rendimiento al algoritmo anterior, ni siquiera si lo que queremos es darle más importancia al recall o la precisión de una de las dos clases. Por eso el *accuracy* global se ve afectado bajando a un 82 %. Los resultados son muy buenos, aún así, habiendo obtenido mejores resultados en todas las métricas, el otro algoritmo es la mejor opción hasta el momento para este conjunto de datos. Esto se puede ver en la matriz de confusión, se observa como la clase 1 ha obtenido más Falsos Negativos que la clase 2. Por otro lado, es importante saber que, al tener tan pocas instancias en test, no se puede saber si este análisis es realmente concluyente. Puede haber sido una casualidad de la separación del conjunto de datos en train y test. Esto necesitaría un estudio más exhaustivo de la distribución del conjunto de datos.

A continuación el *clasification report* del algoritmo SVC:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.80	0.89	0.84	9
1	0.86	0.75	0.80	8
Accuracy		0.82 (17)		
Macro Avg		0.83 (0.82) (0.82) (17)		
Weighted Avg		0.83 (0.82) (0.82) (17)		

Cuadro 21: Classification Report SVC

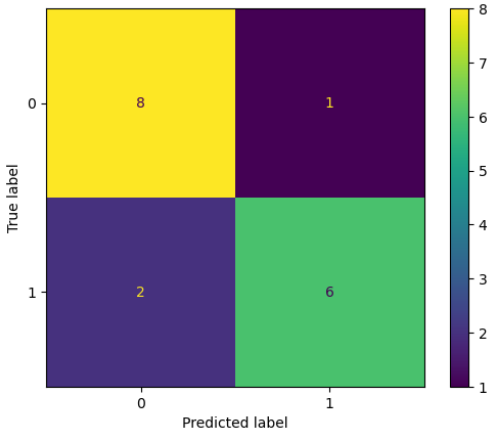


Figura 10: Matriz Confusión SVC

En este caso los resultados en cuanto a métricas se refieren son idénticos al algoritmo anterior de Naive Bayes. Así que el análisis a realizar es el mismo. Los resultados son muy buenos, pero habiendo obtenido unos resultados mejores, es mejor determinar que el mejor algoritmo en un principio para este conjunto de datos es el de Random Forest.

En términos globales se podría decir que, de los algoritmos explorados, el algoritmo con mejores resultados es el Random Forest. Sin embargo, este conjunto de datos tenía muy pocas instancias y no se puede saber con seguridad si los resultados obtenidos son concluyentes. Quizás sería necesario realizar un análisis más profundo analizando la composición del conjunto de datos y si quizás aplicando un *k-folds* se podría intentar mejorar la indepencia de los subconjuntos de datos. Para así obtener resultados más fiables.

3.6. Conclusiones

Para este último problema se han utilizado 2 bases de datos diferentes. Una con muchos datos y desbalanceada y otra con menos datos pero balanceada. Estas bases de datos se tuvieron que importar al servidor MongoDB a través de comandos desde la terminal y los comandos y código que la librería de MongoDB en python facilita.

Ha sido un enfoque interesante para aprender cómo conseguir almacenar grandes conjuntos de datos y disponer de ellos en un servidor en la nube para luego poder utilizarlos desde una interfaz gráfica con la que poder analizar estos datos en más detalle. No sólo con análisis sencillos como los realizados en la Fase 1 de este trabajo en los que se puede visualizar con líneas de código de MongoDB sino con problemas más sencillos en los que se aplican a los datos algoritmos de clasificación.

Para esta parte se han dispuesto además de 2 conjuntos de datos muy interesantes, uno de ellos compuesto de una buena cantidad de datos pero que, sin embargo, las clases se encontraban desbalanceadas. Esto hacía prever un problema en el rendimieto del modelo, pero al tener unn gran número de instancias no se esperaban resultados también desbalanceados. Por otro lado, el segundo conjunto de datos no tenía una gran cantidad de datos, pero estos estaban balanceados. Se previó en un primer momento el problema al que esto podía llevar. Sin embargo, el hecho de que los datos estuvieran balanceados ha dado lugar a obtener resultados muy positivos.

Como análisis más profundo del impacto de los conjuntos de datos en el rendimiento de los modelos, hay que tener en cuenta la naturaleza de los conjuntos de datos. No sólo el número de atributos y de clases, sino también cómo luego se distribuyen estos datos en los subconjuntos de entrenamiento y test. Una mala distribución puede llevar a obtener y posteriormente hacer un análisis demasiado positivo o demasiado negativo de los resultados.

En resumen, ha sido un trabajo interesante para explorar las posibilidades que ofrecen herramientas de almacenamiento de datos como MongoDB y el potencial que alcanza al utilizar a su vez otras herramientas y técnicas como, en este caso ha sido, la clasificación de estos datos.