



UNIVERSIDAD
DE
CÓRDOBA

Trabajo 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

Carlos Checa Moreno

10 de marzo de 2025



ÍNDICE GENERAL

FASE 1	SOLUCIÓN NOSQL MONGODB	2
1.1	OBJETIVO	2
1.2	TRABAJO A REALIZAR	2
1.3	SOLUCIÓN	3
1.3.1	<i>Estructura de colecciones</i>	<i>3</i>
1.3.2	<i>Carga de Información</i>	<i>3</i>
1.3.3	<i>Consultas.....</i>	<i>5</i>
FASE 2	APLICACIONES CIENTÍFICAS EMPRESARIALES (I).....	7
2.1	OBJETIVO	7
2.2	TRABAJO A REALIZAR	7
2.3	SOLUCIÓN	8
2.3.1	<i>Cargar datos.....</i>	<i>8</i>
2.3.2	<i>Preprocesado de datos</i>	<i>9</i>
2.3.3	<i>Implementación modelo de clasificación</i>	<i>10</i>

Fase 1

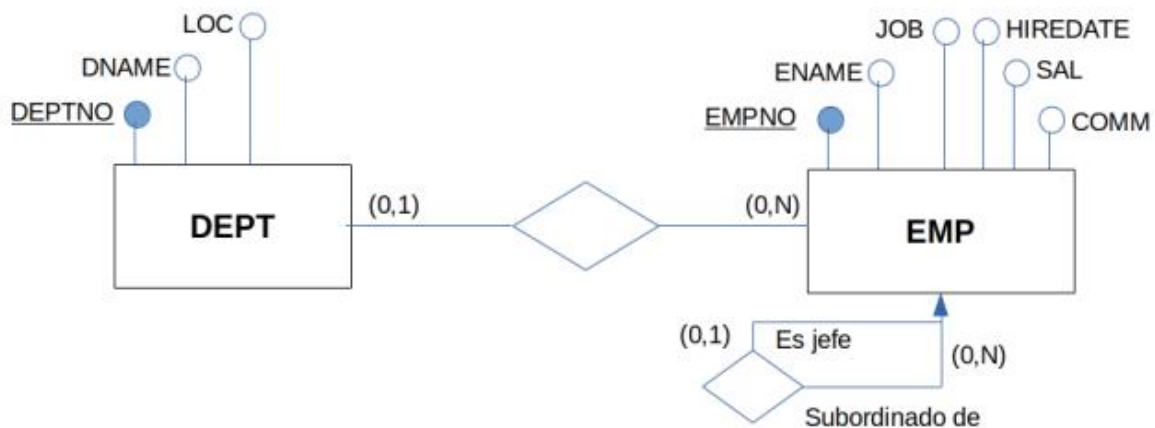
SOLUCIÓN NoSQL MongoDB

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.

1.2 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.

1.3 Solución

1.3.1 Estructura de colecciones

Haré una colección para cada tipo de entidad del esquema, una para DEPT y para EMP.

1.3.1.1 Colección DEPT

```
{
  "_id": ObjectId("..."),
  "DEPTNO": "Identificador del departamento",
  "DNAME": "Título del departamento",
  "LOC": "Ubicación del departamento",
  "MANAGER": "Número de empleado (EMPNO) que lidera el departamento"
}
```

1.3.1.2 Colección EMP

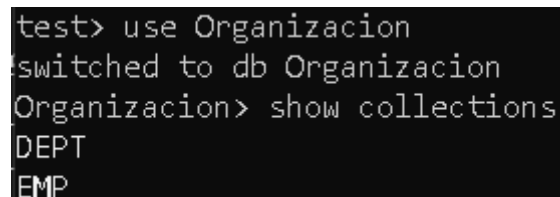
```
{
  "_id": ObjectId("..."),
  "EMPNO": "Número identificador del empleado",
  "ENAME": "Nombre completo del empleado",
  "JOB": "Cargo que desempeña el empleado",
  "HIREDATE": "Fecha en la que el empleado fue contratado",
  "SAL": "Ingreso salarial del empleado",
  "COMM": "Bonificación o comisión del empleado",
  "DEPTNO": "Número identificador del departamento al que pertenece",
  "ISJEFE": "Indica si el empleado es el jefe",
  "JEFE": "Si no es jefe, identifica a su jefe "
}
```

1.3.2 Carga de Información

```
from pymongo import MongoClient

cliente = MongoClient("mongodb://localhost:27017/")
db = cliente["Organizacion"]
colecciones = ["DEPT", "EMP"]

for coleccion in colecciones:
    db.create_collection(coleccion)
```



```
test> use Organizacion
switched to db Organizacion
Organizacion> show collections
DEPT
EMP
```

Figura 1: Colecciones creadas

```
from pymongo import MongoClient
from bson.objectid import ObjectId
from datetime import datetime
```

```

# Conexión a MongoDB
cliente = MongoClient("mongodb://localhost:27017/")
db = cliente["Organizacion"]

db.DEPT.insert_many([
    {"_id": ObjectId(), "DEPTNO": 10, "DNAME": "Contabilidad", "LOC": "New
York", "MANAGER": 7839},
    {"_id": ObjectId(), "DEPTNO": 20, "DNAME": "Investigación", "LOC":
"Dallas", "MANAGER": 7566},
    {"_id": ObjectId(), "DEPTNO": 30, "DNAME": "Ventas", "LOC": "Chicago",
"MANAGER": 7698},
    {"_id": ObjectId(), "DEPTNO": 40, "DNAME": "Operaciones", "LOC":
"Boston", "MANAGER": 7782},
    {"_id": ObjectId(), "DEPTNO": 50, "DNAME": "Recursos Humanos", "LOC":
"Seattle", "MANAGER": 7902},
    {"_id": ObjectId(), "DEPTNO": 60, "DNAME": "Legal", "LOC": "San
Francisco", "MANAGER": 7654},
    {"_id": ObjectId(), "DEPTNO": 70, "DNAME": "Marketing", "LOC": "Los
Angeles", "MANAGER": 7499},
    {"_id": ObjectId(), "DEPTNO": 80, "DNAME": "IT", "LOC": "Miami",
"MANAGER": 7788},
    {"_id": ObjectId(), "DEPTNO": 90, "DNAME": "Producción", "LOC":
"Denver", "MANAGER": 7521},
    {"_id": ObjectId(), "DEPTNO": 100, "DNAME": "Compras", "LOC":
"Houston", "MANAGER": 7844},
    {"_id": ObjectId(), "DEPTNO": 110, "DNAME": "Calidad", "LOC": "Austin",
"MANAGER": 7934},
    {"_id": ObjectId(), "DEPTNO": 120, "DNAME": "Logística", "LOC":
"Phoenix", "MANAGER": 7900},
    {"_id": ObjectId(), "DEPTNO": 130, "DNAME": "Relaciones Públicas",
"LOC": "Las Vegas", "MANAGER": 7782},
    {"_id": ObjectId(), "DEPTNO": 140, "DNAME": "Desarrollo", "LOC": "San
Diego", "MANAGER": 7839},
    {"_id": ObjectId(), "DEPTNO": 150, "DNAME": "Soporte", "LOC":
"Philadelphia", "MANAGER": 7566},
    {"_id": ObjectId(), "DEPTNO": 160, "DNAME": "Finanzas", "LOC":
"Portland", "MANAGER": 7698},
    {"_id": ObjectId(), "DEPTNO": 170, "DNAME": "Proyectos", "LOC":
"Detroit", "MANAGER": 7782},
    {"_id": ObjectId(), "DEPTNO": 180, "DNAME": "Ingeniería", "LOC":
"Columbus", "MANAGER": 7839},
    {"_id": ObjectId(), "DEPTNO": 190, "DNAME": "Seguridad", "LOC":
"Charlotte", "MANAGER": 7566},
    {"_id": ObjectId(), "DEPTNO": 200, "DNAME": "Investigación y
Desarrollo", "LOC": "Indianapolis", "MANAGER": 7698}
])

db.EMP.insert_many([
    {"_id": ObjectId(), "EMPNO": 7839, "ENAME": "Rey", "JOB": "Presidente",
"HIREDATE": datetime(2020, 6, 9), "SAL": 5000, "COMM": None, "DEPTNO": 10,
"ISJEFE": True, "JEFE": None},
    {"_id": ObjectId(), "EMPNO": 7566, "ENAME": "Martinez", "JOB":
"Gerente", "HIREDATE": datetime(2021, 5, 19), "SAL": 3500, "COMM": None,
"DEPTNO": 20, "ISJEFE": True, "JEFE": 7839},
    {"_id": ObjectId(), "EMPNO": 7698, "ENAME": "Gomez", "JOB": "Gerente",
"HIREDATE": datetime(2022, 3, 15), "SAL": 3200, "COMM": None, "DEPTNO": 30,
"ISJEFE": True, "JEFE": 7839},
    {"_id": ObjectId(), "EMPNO": 7782, "ENAME": "Perez", "JOB": "Gerente",
"HIREDATE": datetime(2022, 8, 5), "SAL": 3100, "COMM": None, "DEPTNO": 40,
"ISJEFE": True, "JEFE": 7839},

```

```
{ "_id": ObjectId(), "EMPNO": 7844, "ENAME": "Lopez", "JOB": "Vendedor",
"HIREDATE": datetime(2023, 1, 20), "SAL": 1500, "COMM": 300, "DEPTNO": 30,
"ISJEFE": False, "JEFE": 7698},
{ "_id": ObjectId(), "EMPNO": 7876, "ENAME": "Rodriguez", "JOB":
"Analista", "HIREDATE": datetime(2023, 4, 10), "SAL": 2800, "COMM": None,
"DEPTNO": 20, "ISJEFE": False, "JEFE": 7566}
]
```

1.3.3 Consultas

1.3.3.1 Consulta 1: Número empleados

El departamento con más empleados es: 20 con 2 empleados.

```
pipeline = [
    {"$group": {"_id": "$DEPTNO", "total_empleados": {"$sum": 1}}},
    {"$sort": {"total_empleados": -1}},
    {"$limit": 1}
]

resultado = list(db.EMP.aggregate(pipeline))
print(f"El departamento con más empleados es: {resultado[0]['_id']} con
{resultado[0]['total_empleados']} empleados.")
```

1.3.3.2 Consulta 2: Salario máximo

El empleado con el salario más alto es Rey con un salario de 5000.

```
pipeline = [
    {"$sort": {"SAL": -1}},
    {"$limit": 1}
]

resultado = list(db.EMP.aggregate(pipeline))
if resultado:
    empleado = resultado[0]
    print(f"El empleado con el salario más alto es {empleado['ENAME']} con
un salario de {empleado['SAL']}.")
```

1.3.3.3 Consulta 3: Número empleados por departamentos

Número de empleados por departamento:

- Departamento 20: 2 empleados
- Departamento 30: 2 empleados
- Departamento 10: 1 empleados
- Departamento 40: 1 empleados

```
pipeline = [
    {"$group": {"_id": "$DEPTNO", "total_empleados": {"$sum": 1}}},
```

```
        {"$sort": {"total_empleados": -1}}
    ]

resultado = list(db.EMP.aggregate(pipeline))

print("Número de empleados por departamento:")
for departamento in resultado:
    print(f"Departamento {departamento['_id']}:
    {departamento['total_empleados']} empleados")
```

Fase 2

APLICACIONES CIENTÍFICAS EMPRESARIALES (I)

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.

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 “1”; Inactive ”0”). 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.

2.3 Solución

2.3.1 Cargar datos

Primero, cargaré las bases de datos:

```
import seaborn as sns

client = MongoClient("mongodb://localhost:27017/");
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('CDS29');
db_CDS16=load_database('CDS16');

print("\nCDS29: ");
print("- ", db_CDS29.list_collection_names());
print("- ", db_CDS29.CDS29.molecules);
print("- ", db_CDS29.CDS29.mfp_counts);

print("\nCDS16");
print("- ", db_CDS16.list_collection_names());
print("- ", db_CDS16.CDS29.molecules);
print("- ", db_CDS16.CDS29.mfp_counts);
```

Obtenemos la siguiente salida:

```

CDS29:
- ['molecules', 'mfp_counts']
- Collection(Database(MongoClient(host=['localhost:27017'],
document_class=dict, tz_aware=False, connect=True), 'CDS29'),
'CDS29.molecules')
- Collection(Database(MongoClient(host=['localhost:27017'],
document_class=dict, tz_aware=False, connect=True), 'CDS29'),
'CDS29.mfp_counts')

CDS16:
- ['mfp_counts', 'molecules']
- Collection(Database(MongoClient(host=['localhost:27017'],
document_class=dict, tz_aware=False, connect=True), 'CDS16'),
'CDS29.molecules')
- Collection(Database(MongoClient(host=['localhost:27017'],
document_class=dict, tz_aware=False, connect=True), 'CDS16'),
'CDS29.mfp_counts')

```

Y cargo los datos de las moléculas de CDS16 y CDS29.

```

import pandas as pd

def load_data(db, collection_name):
    collection = db[collection_name]
    cursor = collection.find()
    data = pd.DataFrame(list(cursor))
    return data

data_CDS29 = load_data(db_CDS29, 'molecules')
data_CDS16 = load_data(db_CDS16, 'molecules')

```

_id	smiles	class	MOLECULEID	rdmol	mfp
0 1	Nc1nc2cnc2C2OC(CO)C(O)C2O)c1[nH]1	Inactive	M18965	b'\xef\xbe\xad\xde\x00\x00\x00\x0b\x00\x00...	['bits': [71, 75, 80, 90, 128, 147, 152, 194, ...
1 2.0	Cc1cn(-c2cc(NC(=O)c3ccc(C)c(Nc4ncac(-c5ccncc5))...	Inactive	M413388	b'\xef\xbe\xad\xde\x00\x00\x00\x0b\x00\x00...	['bits': [19, 33, 61, 64, 84, 90, 114, 128, 13...
2 3.0	O=C(O)CC(O)CC(O)C=Cc1dC2CC2)nc2ccccc2c1-c1ccc...	Inactive	M3412451	b'\xef\xbe\xad\xde\x00\x00\x00\x0b\x00\x00...	['bits': [1, 25, 64, 73, 80, 90, 105, 117, 136...
3 4.0	O=C1CC(O)CC(C=Cc2c(C3CC3)nc3ccccc3c2-c2ccc(F)c...	Inactive	M97340488	b'\xef\xbe\xad\xde\x00\x00\x00\x0b\x00\x00...	['bits': [64, 73, 80, 90, 105, 136, 175, 204, ...
4 5.0	Cc1nnq(C(=O)NC(CO)C2nc(C(=O)NCc3ccc(F)cc3)c1...	Inactive	M409346	b'\xef\xbe\xad\xde\x00\x00\x00\x0b\x00\x00...	['bits': [26, 33, 80, 90, 114, 121, 128, 140, ...

Figura 2: Head de data_CDS16

_id	smiles	class	MOLECULEID	rdmol	mfp
0 mol1382	COc1cc(-c2cc(C(=O)Nc3nc4ccccc4s3)c3ccccc3n2)cc...	0	M609802	b'\xef\xbe\xad\xde\x00\x00\x00\x0b\x00\x00...	['bits': [33, 55, 64, 73, 118, 128, 136, 156, ...
1 mol1383	Cc1cc(O)c(N/C=C2/C(=O)N(c3ncsc3)C(=O)c3ccccc32...	0	M753571	b'\xef\xbe\xad\xde\x00\x00\x00\x0b\x00\x00...	['bits': [14, 33, 53, 64, 96, 128, 175, 202, 2...
2 mol1384	O=C(COC(=O)c1sc2ccccc2c1C)NC1CCCCC1	0	M754020	b'\xef\xbe\xad\xde\x00\x00\x00\x0b\x00\x00...	['bits': [2, 4, 41, 64, 80, 86, 128, 145, 147, ...
3 mol1385	CC(Oc1ccc2oc3ccccc3c2c1)C(=O)NC12CC3CC(C(C3)C...	0	M753862	b'\xef\xbe\xad\xde\x00\x00\x00\x0b\x00\x00...	['bits': [1, 33, 35, 36, 60, 64, 104, 109, 128...
4 mol1386	COc1cc(C=C(C\#N)C(=O)NC2CCCCC2)ccc1OS(=O)=O...	0	M753662	b'\xef\xbe\xad\xde\x00\x00\x00\x0b\x00\x00...	['bits': [4, 25, 33, 55, 94, 128, 130, 179, 18...

Figura 3: Head de data_CDS19

2.3.2 Preprocesado de datos

Aplicaré un RandomForest. Para esta tarea convierto mfp en una matriz binaria para poder utilizarla como características de nuestro modelo.

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

```

```

from sklearn.metrics import accuracy_score, roc_auc_score,
cohen_kappa_score, balanced_accuracy_score

# Función para transformar los fingerprints en un DataFrame binario
def transform_fingerprints(data, num_bits=1024):
    fingerprints = np.zeros((len(data), num_bits))

    for i, fp in enumerate(data['mfp']):
        for bit in fp['bits']:
            if bit < num_bits:
                fingerprints[i, bit] = 1

    return pd.DataFrame(fingerprints, columns=[f'FP{i}' for i in
range(num_bits)])

for dataset, name in zip([data_CDS29, data_CDS16], ['CDS29', 'CDS16']):
    print(f'Procesando dataset {name}')

    X = transform_fingerprints(dataset)

    # Convertir la variable de respuesta a binario
    dataset['class'] = dataset['class'].map({'Active': 1, 'Inactive': 0})
    y = dataset['class']

    # Eliminar columnas con solo ceros
    X = X.loc[:, (X != 0).any(axis=0)]

    # 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)

```

2.3.3 Implementación modelo de clasificación

```

# Entrenar con Random Forest
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predicciones
y_pred = clf.predict(X_test)
y_prob = clf.predict_proba(X_test)[:, 1]

# Evaluación
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_prob)
kappa = cohen_kappa_score(y_test, y_pred)
gmean = balanced_accuracy_score(y_test, y_pred)

# Métricas
print(f'Resultados para {name}:')
print(f'Accuracy: {accuracy:.4f}')
print(f'ROC AUC: {roc_auc:.4f}')
print(f'Cohen Kappa: {kappa:.4f}')
print(f'G-Mean: {gmean:.4f}\n')

```

2.3.3.1 Resultados

Obtengo los siguientes resultados:

- CDS29:
 - Accuracy: 0.8061
 - ROC AUC: 0.7021
 - Cohen Kappa: 0.1418
 - G-Mean: 0.5508
- CDS16:
 - Accuracy: 0.8235
 - ROC AUC: 0.9653
 - Cohen Kappa: 0.6483
 - G-Mean: 0.8264

2.3.3.2 Interpretación

En cuanto al **Accuracy**, CDS16 alcanza un 82.35%, superando el 80.61% de CDS29, lo que indica un mejor rendimiento general del modelo en la predicción de la clase correcta. Sin embargo, la diferencia más significativa se observa en la métrica **ROC AUC**, donde CDS16 obtiene un valor de 0.9653 frente a 0.7021 en CDS29. Esto sugiere que el modelo en CDS16 es mucho más efectivo en la discriminación entre clases activas e inactivas.

Cohen Kappa ajusta el efecto del azar en la proporción de la concordancia observada. En CDS29, el valor de 0.1418 indica una concordancia baja. Por otra parte, en CDS16 tenemos 0.6483 que refleja una concordancia moderada, sugiriendo un mejor desempeño del modelo en CDS16.

G-Mean evalúa el equilibrio entre la sensibilidad y la especificidad, útil en conjuntos de datos desbalanceados. En CDS29, el valor de 0.5508 sugiere dificultades en la clasificación de ambas clases, mientras que en CDS16, el 0.8264 indica que el modelo distingue mejor entre compuestos activos e inactivos.
