



Universidad de Córdoba

MÁSTER EN INTELIGENCIA COMPUTACIONAL E INTERNET DE LAS COSAS

SOLUCIÓN NOSQL MONGODB CARACTERIZACIÓN Y PREDICCIÓN DE ACTIVIDADES/PROPIEDADES DE COMPUESTOS (DISEÑO DE FÁRMACOS)

Autor: Alberto Fernández Merchán

Profesores:

Gonzalo Cerruela García
Domingo Ortíz Boyer
Juan A. Romero del Castillo

Asignatura: Análisis, Diseño y Procesamiento de los datos
2025

Índice

1. FASE 1: SOLUCIÓN NOSQL MONGODB	2
1.1. Objetivo	2
1.2. Trabajo a realizar	2
1.3. Estructura de colecciones de MongoDB	2
1.3.1. Colección de Departamentos	3
1.3.2. Colección de Empleados	3
1.4. Carga de datos	4
1.4.1. Colección de Departamentos	4
1.4.2. Colección de Empleados	4
1.4.3. Consultas	6
2. FASE 2: APLICACIONES CIENTÍFICAS Y EMPRESARIALES (I)	9
2.1. Objetivo	9
2.2. Trabajo a realizar	9
2.3. Carga de datos	9
2.4. Preprocesamiento	9
2.5. Clasificador	11
2.5.1. Random Forest	11
2.5.2. Support Vector Machine	11
2.5.3. Multilayer Perceptron	12
2.6. Análisis de resultados	13
2.6.1. Random Forest	13
2.6.2. Support Vector Machine (SVM)	14
2.6.3. Multilayer Perceptron (MLP)	15
2.7. Conclusiones	16

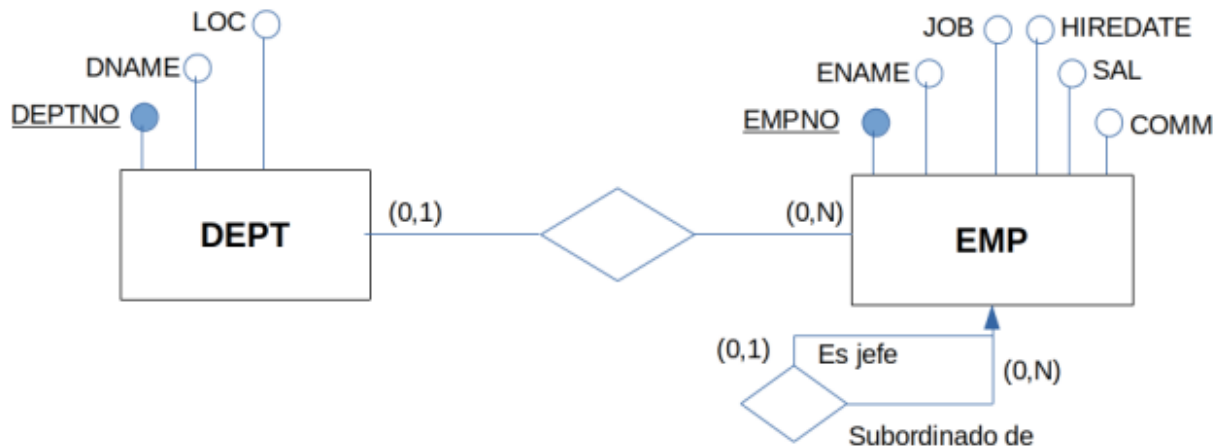
1. FASE 1: SOLUCIÓN NOSQL MONGODB

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. Estructura de colecciones de MongoDB

Para diseñar una estructura de colecciones de documentos con MongoDB podríamos utilizar el siguiente enfoque:

- Departamentos: Contendría los atributos de los departamentos:
 - *_id*: el identificador del objeto.
 - DEPTNO: Número de departamento.
 - DNAME: Nombre del departamento.
 - LOC: Localización del departamento.
- Empleados:
 - *_id*: el identificador del objeto.
 - EMPNO: Número de empleado.
 - ENAME: Nombre del empleado.
 - JOB: Trabajo del empleado.
 - HIREDATE: Fecha de contratación.

- SAL: Salario del empleado.
- COMM: Comisión del empleado.
- DEPTNO: Número de departamento al que pertenece el empleado.
- ISBOSS: Si es jefe.
- BOSS: De quién es subordinado.

Para crear las colecciones utilizamos:

```

1 {
2   const database = 'EJERCICIO_1';
3   const collection1 = 'DEPT';
4   const collection2 = 'EMP';
5
6   // Seleccionar (o crear) la base de datos
7   use(database);
8
9   // Crear las colecciones
10  db.createCollection(collection1);
11  db.createCollection(collection2);
12 }

```

1.3.1. Colección de Departamentos

En la colección de Departamentos tendremos los siguientes atributos:

```

1 {
2   "_id": "ObjectId()", // Identificador único en MongoDB
3   "DEPTNO": "Number",  // Número de departamento (único)
4   "DNAME": "String",   // Nombre del departamento
5   "LOC": "String"      // Localización del departamento
6 }

```

1.3.2. Colección de Empleados

En la colección Empleados, tendremos los siguientes atributos:

```

1 {
2   "_id": "ObjectId()", // Identificador único en MongoDB
3   "EMPNO": "Number",   // Número de empleado (único)
4   "ENAME": "String",   // Nombre del empleado
5   "JOB": "String",     // Puesto de trabajo
6   "HIREDATE": "Date",  // Fecha de contratación
7   "SAL": "Number",     // Salario
8   "COMM": "Number",    // Comisión (puede ser nula)
9   "DEPTNO": "Number",  // Número de departamento al que pertenece (clave foránea)
10  "JEFE": "Number",    // Número de empleado del jefe (puede ser nulo)
11  "SUBORDINADOS": "[Number]" // Lista de EMPNO de subordinados
12 }
13

```

1.4. Carga de datos

Para cargar datos en las colecciones utilizaremos el siguiente script:

1.4.1. Colección de Departamentos

```
1 // Insertar datos en la colección DEPT (Departamentos)
2 db[collection1].insertMany([
3   {
4     "DEPTNO": 10,
5     "DNAME": "Contabilidad",
6     "LOC": "Madrid"
7   },
8   {
9     "DEPTNO": 20,
10    "DNAME": "Ventas",
11    "LOC": "Barcelona"
12  },
13  {
14    "DEPTNO": 30,
15    "DNAME": "Investigación",
16    "LOC": "Valencia"
17  }
18 ]);
```

1.4.2. Colección de Empleados

```
1 // Insertar datos en la colección EMP (Empleados)
2 db[collection2].insertMany([
3   {
4     "EMPNO": 1001,
5     "ENAME": "Juan Pérez",
6     "JOB": "Analista",
7     "HIREDATE": new Date("2023-03-15"),
8     "SAL": 2500,
9     "COMM": 500,
10    "DEPTNO": 10,
11    "JEFE": null,
12    "SUBORDINADOS": [1002, 1003]
13  },
14  {
15    "EMPNO": 1002,
16    "ENAME": "Ana Gómez",
17    "JOB": "Desarrolladora",
18    "HIREDATE": new Date("2022-06-20"),
19    "SAL": 2200,
20    "COMM": 300,
21    "DEPTNO": 10,
22    "JEFE": 1001,
23    "SUBORDINADOS": []
24  },
25  {
26    "EMPNO": 1003,
27    "ENAME": "Carlos Ruiz",
28    "JOB": "Técnico",
29    "HIREDATE": new Date("2021-09-10"),
30    "SAL": 1800,
31    "COMM": null,
32    "DEPTNO": 10,
33    "JEFE": 1001,
```

```

34     "SUBORDINADOS": []
35 },
36 {
37     "EMPNO": 2001,
38     "ENAME": "Sofia López",
39     "JOB": "Vendedora",
40     "HIREDATE": new Date("2022-12-01"),
41     "SAL": 2000,
42     "COMM": 700,
43     "DEPTNO": 20,
44     "JEFE": null,
45     "SUBORDINADOS": []
46 }
47 ]);

```

Comprobamos que se ha creado correctamente:

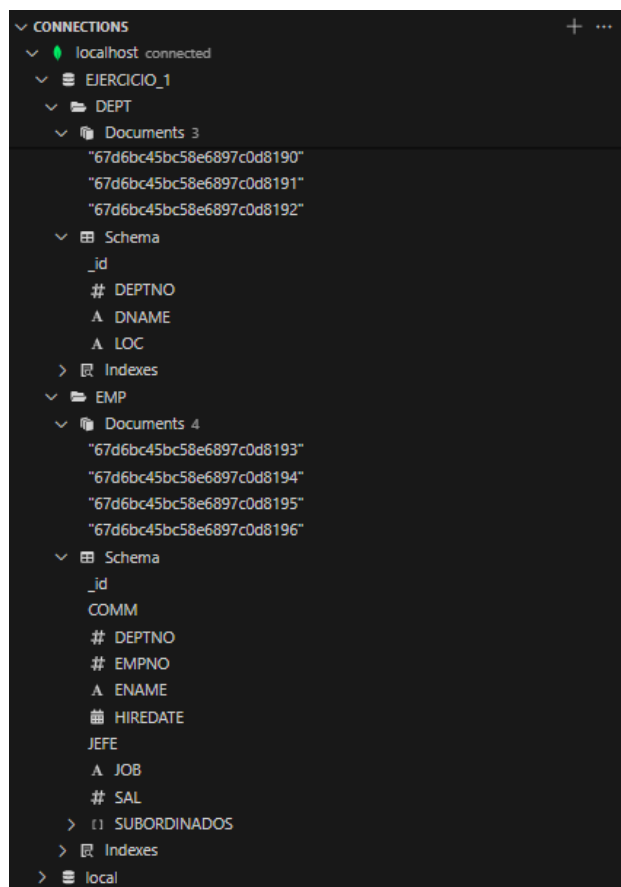
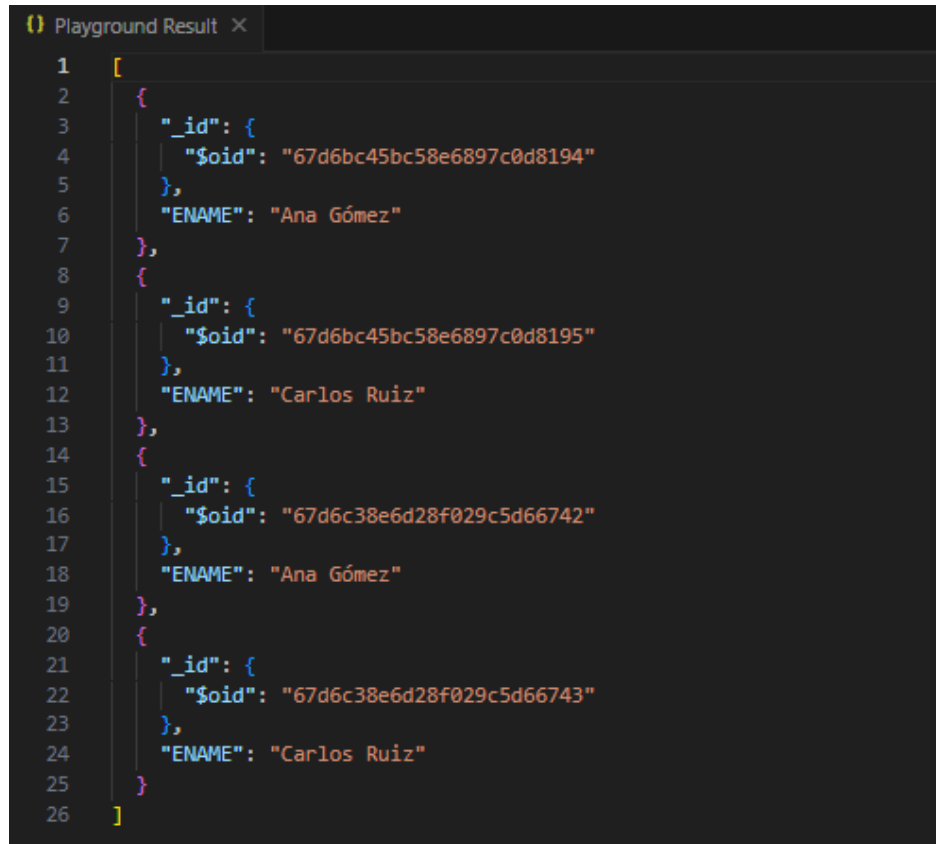


Figura 1: Esquema de la base de datos

1.4.3. Consultas

1. Muestra los nombres de los empleados subordinados del empleado 1001:

```
1 db.EMP.find({ "JEFE": 1001},  
2   {"ENAME": 1}  
3   ).pretty();
```



```
{  
  "_id": {  
    "$oid": "67d6bc45bc58e6897c0d8194"  
  },  
  "ENAME": "Ana Gómez"  
},  
{  
  "_id": {  
    "$oid": "67d6bc45bc58e6897c0d8195"  
  },  
  "ENAME": "Carlos Ruiz"  
},  
{  
  "_id": {  
    "$oid": "67d6c38e6d28f029c5d66742"  
  },  
  "ENAME": "Ana Gómez"  
},  
{  
  "_id": {  
    "$oid": "67d6c38e6d28f029c5d66743"  
  },  
  "ENAME": "Carlos Ruiz"  
}
```

2. Obtener los empleados con salario mayor a 2000:

```
1 db.EMP.find({ SAL: { $gt: 2000 } })
```

```
Playground Result X
1  [
2    {
3      "_id": {
4        "$oid": "67d6c38e6d28f029c5d66741"
5      },
6      "EMPNO": 1001,
7      "ENAME": "Juan Pérez",
8      "JOB": "Analista",
9      "HIREDATE": {
10       "$date": "2023-03-15T00:00:00Z"
11     },
12     "SAL": 2500,
13     "COMM": 500,
14     "DEPTNO": 10,
15     "JEFE": null,
16     "SUBORDINADOS": [
17       1002,
18       1003
19     ]
20   },
21   {
22     "_id": {
23       "$oid": "67d6c38e6d28f029c5d66742"
24     },
25     "EMPNO": 1002,
26     "ENAME": "Ana Gómez",
27     "JOB": "Desarrolladora",
28     "HIREDATE": {
29       "$date": "2022-06-20T00:00:00Z"
30     },
31     "SAL": 2200,
32     "COMM": 300,
33     "DEPTNO": 10,
34     "JEFE": 1001,
35     "SUBORDINADOS": []
36   }
37 ]
```


3. Obtener los empleados que trabajan en Contabilidad:

```
1      db.EMP.aggregate([
2          {
3              $lookup: {
4                  from: "DEPT",
5                  localField: "DEPTNO",
6                  foreignField: "DEPTNO",
7                  as: "DEPT"
8              }
9          },
10         {
11             $match: { "DEPT.DNAME": "Contabilidad" }
12         },
13         {
14             $project: {
15                 ENAME: 1,
16                 "DEPT.DNAME": 1
17             }
18         }
19     ])
```

```
[
  {
    "_id": {
      "$oid": "67d6c38e6d28f029c5d66741"
    },
    "ENAME": "Juan Pérez",
    "DEPT": [
      {
        "DNAME": "Contabilidad"
      }
    ]
  },
  {
    "_id": {
      "$oid": "67d6c38e6d28f029c5d66742"
    },
    "ENAME": "Ana Gómez",
    "DEPT": [
      {
        "DNAME": "Contabilidad"
      }
    ]
  },
  {
    "_id": {
      "$oid": "67d6c38e6d28f029c5d66743"
    },
    "ENAME": "Carlos Ruiz",
    "DEPT": [
      {
        "DNAME": "Contabilidad"
      }
    ]
  }
]
```

2. FASE 2: APLICACIONES CIENTÍFICAS Y 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, debemos implementar el código python necesario para construir un clasificador que prediga el tipo de actividad biológica de cada fármaco (*Active* o *Inactive*).

2.3. Carga de datos

Para cargar los datos utilizaremos la librería de pyMongo encapsulada en la función mostrada en la Figura 2.

```
import pymongo

def connect(database="Milan_CDR_db", collection="Milan_CDR_c"):

    client = pymongo.MongoClient("mongodb://afmhuelva:3NZmlzuSchh9J6k4@localhost:27017/") # Cambia la URL s
    db = client[database] # Nombre de la base de datos
    collection = db[collection] # Nombre de la colección dentro de la base de datos

    return client, db, collection
```

Figura 2: Código para conectarse a la base de datos

A continuación, descargamos las colecciones de la base de datos CDS16 y CDS29 y las encapsulamos en un DataFrame de la biblioteca de Pandas, como se puede observar en la Figura 3

```
if __name__ == "__main__":

    # Preprocesamiento de datos:
    client, db, collection_CDS16 = dc.connect("CDS16", "molecules")
    client, db, collection_CDS29 = dc.connect("CDS29", "molecules")
    data_CDS16 = pd.DataFrame(list(collection_CDS16.find()))
    data_CDS29 = pd.DataFrame(list(collection_CDS29.find()))
```

Figura 3: Encapsulación de las bases de datos en un DataFrame

2.4. Preprocesamiento

Antes de empezar a clasificar los datos, primero debemos expandir la huella de los compuestos químicos. Utilizaremos la función *expand fingerprint* que se ha creado para escribir un '1' donde marque la huella molecular (Figura 4).

Tras expandir la huella, procedemos a dividir el conjunto de datos en variables predictoras (X) y la clase a la que pertenece cada instancia (Y) como se puede observar en el código de la Figura 5.

```
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 # Colocamos un 1 donde nos indique en la mfp
    return fingerprint
```

Figura 4: Funcion *expand fingerprint*

```
# Variables predictorias: mfp (1024)
# Aplicar la función a cada fila
data_CDS29['mfp'] = data_CDS29['mfp'].apply(lambda mfp_data: expand_fingerprint(mfp_data))
data_CDS16['mfp'] = data_CDS16['mfp'].apply(lambda mfp_data: expand_fingerprint(mfp_data))

# Extraemos las variables predictorias:
# Variables predictorias y variable respuesta
X_CDS29 = pd.DataFrame(data_CDS29['mfp'].tolist())
X_CDS16 = pd.DataFrame(data_CDS16['mfp'].tolist())

y_CDS29 = pd.DataFrame(data_CDS29['class'].map({'Active':True, 'Inactive':False})) # Convertimos a
y_CDS16 = pd.DataFrame(data_CDS16['class'].map({'Active':True, 'Inactive':False}))
```

Figura 5: División entre variables predictorias y la clase.

Observamos que la colección **CDS16** tiene 41 ejemplos de cada una de las clases, es decir, es un dataset balanceado. Sin embargo, **CDS29** tiene más ejemplos negativos que positivos, está desbalanceado.

A la hora de dividir cada dataset en conjuntos de entrenamiento y test, debemos tener en cuenta este desbalanceo de clases (ver Figura 6). Para dividir los conjuntos, utilizaremos la funcion *train_test_split()* utilizando la opcion de estratificar (*stratify*) en el caso del dataset desbalanceado.

```
# Dividir en conjunto de entrenamiento y prueba con estratificación solo en el dataset desbalanceado
X_train_CDS29, X_test_CDS29, y_train_CDS29, y_test_CDS29 = train_test_split(X_CDS29, y_CDS29, test_size=0.2, random_state=42, stratify=y_CDS29)
X_train_CDS16, X_test_CDS16, y_train_CDS16, y_test_CDS16 = train_test_split(X_CDS16, y_CDS16, test_size=0.2, random_state=42)
```

Figura 6: División de clases en ambos datasets.

2.5. Clasificador

En esta sección se presentan los resultados de los clasificadores evaluados en los conjuntos de datos CDS16 (balanceado) y CDS29 (desbalanceado).

2.5.1. Random Forest

CDS16 : Para el dataset de CDS16 (Tabla 1).

Random Forest - CDS16	
Métrica	Valor
Accuracy	0.76
Kappa	0.52
ROC AUC	0.92
F1 Score	0.71

Cuadro 1: Métricas obtenidas del clasificador basado en Random Forest para CDS16.

CDS29 : Para el dataset de CDS29 (Tabla 2).

Random Forest - CDS29	
Métrica	Valor
Accuracy	0.80
Kappa	0.01
ROC AUC	0.70
F1 Score	0.01

Cuadro 2: Métricas obtenidas del clasificador basado en Random Forest para CDS29.

2.5.2. Support Vector Machine

CDS16 : Para el dataset de CDS16 (Tabla 3).

SVM - CDS16	
Métrica	Valor
Accuracy	0.82
Kappa	0.65
ROC AUC	0.92
F1 Score	0.80

Cuadro 3: Métricas obtenidas del clasificador basado en SVM para CDS16.

CDS29 : Para el dataset de CDS29 (Tabla 4).

SVM - CDS29	
Métrica	Valor
Accuracy	0.81
Kappa	0.04
ROC AUC	0.69
F1 Score	0.06

Cuadro 4: Métricas obtenidas del clasificador basado en SVM para CDS29.

2.5.3. Multilayer Perceptron

CDS16 : Para el dataset de CDS16 (Tabla 5).

MLP - CDS16	
Métrica	Valor
Accuracy	0.88
Kappa	0.74
ROC AUC	0.95
F1 Score	0.83

Cuadro 5: Métricas obtenidas del clasificador basado en MLP para CDS16.

CDS29 : Para el dataset de CDS29 (Tabla 6).

MLP - CDS29	
Métrica	Valor
Accuracy	0.72
Kappa	0.12
ROC AUC	0.65
F1 Score	0.29

Cuadro 6: Métricas obtenidas del clasificador basado en MLP para CDS29.

2.6. Análisis de resultados

En esta sección se analiza el rendimiento de cada clasificador en los datasets balanceado (CDS16) y desbalanceado (CDS29). Se comparan las métricas obtenidas para evaluar el impacto del desbalanceo en la capacidad predictiva de los modelos.

2.6.1. Random Forest

CDS16 El clasificador Random Forest muestra buenos resultados en el dataset balanceado CDS16, con un *accuracy* del 76 %, un *kappa* de 0.52 y un *F1-Score* de 0.71. La métrica ROC AUC alcanza un valor de 0.92, lo que indica una **buena capacidad de discriminación entre las clases**.

La matriz de confusión es la que se muestra en la Figura 7.

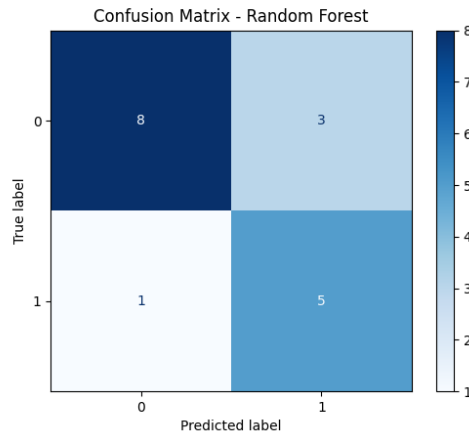


Figura 7: Matriz de confusión del clasificador Random Forest para el dataset CDS16.

CDS29 En el dataset desbalanceado CDS29, el rendimiento de Random Forest disminuye considerablemente. Aunque el *accuracy* es similar (80 %), las métricas *kappa* (0.01) y *F1-Score* (0.01) son extremadamente bajas, lo que indica que el modelo no está clasificando correctamente la clase minoritaria. Además, el ROC AUC cae a 0.70, reflejando una **menor capacidad discriminativa**.

La matriz de confusión es la que se muestra en la Figura 8.

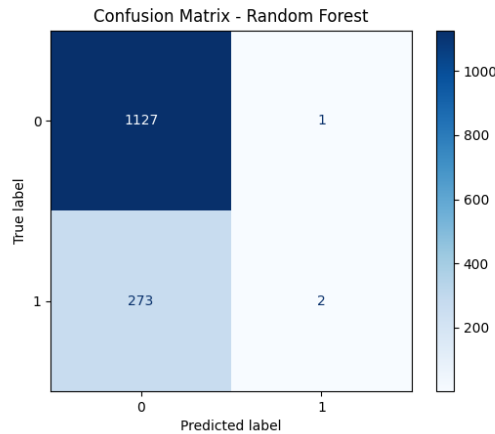


Figura 8: Matriz de confusión del clasificador Random Forest para el dataset CDS29

2.6.2. Support Vector Machine (SVM)

CDS16 El clasificador SVM logra un alto rendimiento en CDS16, con un Accuracy del 82 %, un Kappa de 0.65 y un F1 Score de 0.80. El ROC AUC es de 0.92, lo que demuestra una gran capacidad de separación entre clases.

La matriz de confusión es la que se muestra en la Figura 9.

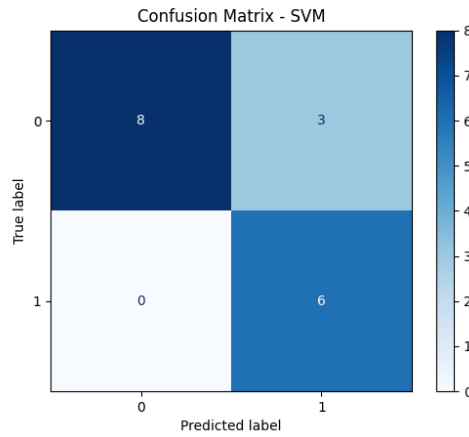


Figura 9: Matriz de confusión del clasificador SVM para el dataset CDS16.

CDS29 En CDS29, las métricas se ven afectadas por el desbalanceo. Aunque el Accuracy sigue siendo relativamente alto (81 %), el Kappa cae a 0.04 y el F1 Score a 0.06, lo que indica un sesgo hacia la clase mayoritaria. El ROC AUC también disminuye a 0.69, evidenciando dificultades en la discriminación entre clases.

La matriz de confusión es la que se muestra en la Figura 10.

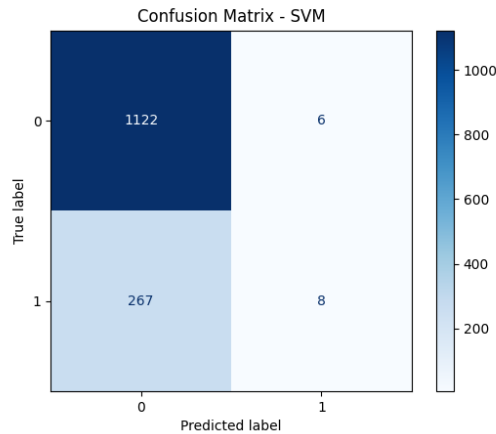


Figura 10: Matriz de confusión del clasificador SVM para el dataset CDS29.

2.6.3. Multilayer Perceptron (MLP)

CDS16 El modelo MLP obtiene los mejores resultados en CDS16, con un Accuracy del 88 %, un Kappa de 0.74 y un F1 Score de 0.83. Además, su ROC AUC es de 0.95, mostrando una gran capacidad para diferenciar las clases.

La matriz de confusión es la que se muestra en la Figura 11.

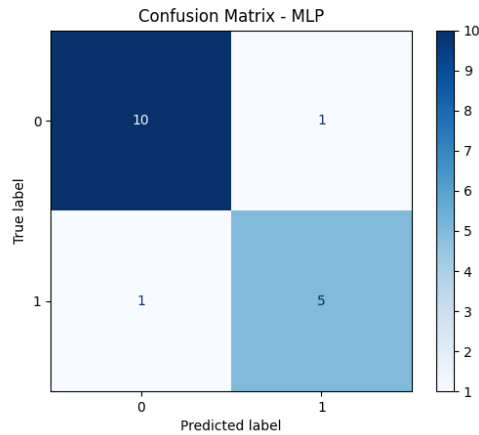


Figura 11: Matriz de confusión del clasificador MLP en el dataset CDS16.

CDS29 En CDS29, el rendimiento del MLP disminuye, pero sigue siendo superior en comparación con los otros clasificadores. El Accuracy cae a 72 %, el Kappa a 0.12 y el F1 Score a 0.29. Aunque estas métricas siguen siendo bajas, el modelo logra una mejor clasificación de la clase minoritaria en comparación con Random Forest y SVM. El ROC AUC es de 0.65, lo que sugiere que el modelo aún tiene dificultades con la discriminación de clases en datos desbalanceados.

La matriz de confusión es la que se muestra en la Figura 12.

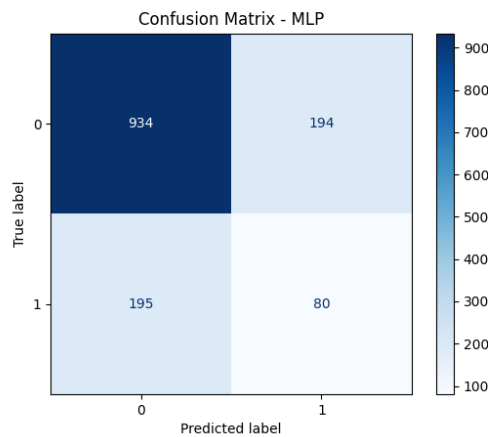


Figura 12: Matriz de confusión del clasificador MLP en el dataset CDS29.

2.7. Conclusiones

Los resultados muestran que todos los clasificadores obtienen un mejor desempeño en el dataset balanceado (CDS16) en comparación con el desbalanceado (CDS29). En CDS29, el Kappa y el F1 Score son muy bajos en todos los modelos, lo que indica que tienen dificultades para clasificar la clase minoritaria. El MLP es más robusto ante el desbalanceo en comparación con Random Forest y SVM, aunque sigue presentando rendimiento bastante bajo.

Estos resultados demuestran la importancia del balanceo de clases en tareas de clasificación, ya que un conjunto de datos desbalanceado puede afectar de forma negativa a la capacidad del modelo para aprender patrones representativos de todas las clases. Para mejorar el rendimiento de los clasificadores en el dataset desbalanceado podría utilizarse alguna de estas soluciones:

1. **Técnicas de Balanceo de Clases:** Aplicar *sobremuestreo de la clase minoritaria* (por ejemplo, SMOTE) o *submuestreo de la clase mayoritaria* para mejorar la distribución de clases en el conjunto de entrenamiento. Esto puede ayudar a los modelos a aprender patrones más representativos de la clase minoritaria.
2. **Análisis de Errores y Ajuste de Umbrales de Decisión:** Realizar un *análisis de los errores de clasificación* para identificar si existen patrones específicos que los modelos no están aprendiendo correctamente, y ajustar los umbrales de decisión de los clasificadores para dar mayor prioridad a la clase minoritaria.
3. **Generación de Datos Sintéticos:** Explorar el uso de *algoritmos generativos*, como los *Generative Adversarial Networks (GANs)*, para crear ejemplos sintéticos de la clase minoritaria y así aumentar la representatividad de todas las clases en el dataset.