Universidad de Córdoba

Máster en Inteligencia Computacional e Internet de las Cosas

MongoDB, Predicción de Actividades y Propiedades de Compuestos

Análisis, Diseño y Procesamiento de Datos Aplicados a las Ciencias y a las Tecnologías (ADP Marzo 2025)

Autor:
Marcos Rivera Gavilán

Profesor:

Gonzalo Cerrueda García

Índice general

Capítulo 1

Ejercicio 1

1.1. Objetivo

El objetivo es afianzar los conocimientos sobre el uso de la base de datos NoSQL MongoDB, así como su modelado y acceso a la información.

1.2. Propuesta de Estructura de Colecciones

A partir del esquema E-R (Departamentos y Empleados, con la relación jefe-subordinado), se pueden diseñar dos colecciones principales en MongoDB:

- **departments**: almacena los datos de cada departamento (deptno, dname, loc, etc.).
- **employees**: almacena los datos de cada empleado (empno, ename, job, hiredate, sal, comm), con referencias a:
  - *deptId*: referencia al departamento correspondiente.
  - *managerId*: referencia al empleado que sea su jefe (opcional o `null` si es presidente).

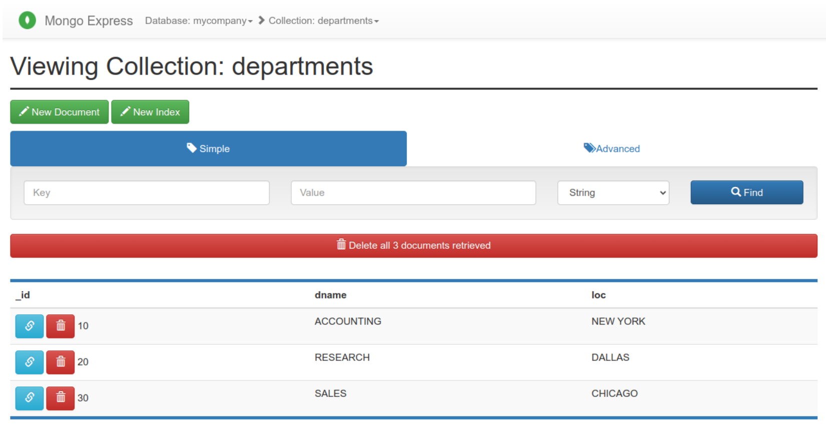

Figura 1.1: Colección `departments` en la base de datos.

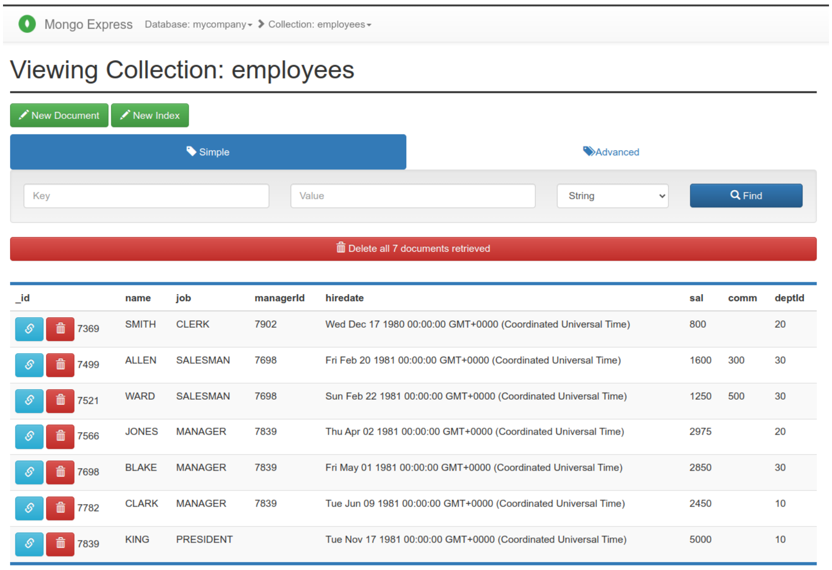

Figura 1.2: Colección `employees` en la base de datos.

Un ejemplo de documento en la colección `departments` podría ser:

Listing 1.1: Ejemplo de documento en `departments`

```
{
  "_id": 10,
  "dname": "ACCOUNTING",
  "loc": "NEW YORK"
}
```

Un ejemplo de documento en la colección `employees` podría ser:

Listing 1.2: Ejemplo de documento en `employees`

```
{
  "_id": 7369,
  "name": "SMITH",
  "job": "CLERK",
  "managerId": 7902,
  "hiredate": ISODate("1980-12-17"),
  "sal": 800,
  "comm": null,
  "deptId": 20
}
```

1.3. Carga de Información de Prueba

Para contar con datos de validación, se pueden realizar inserciones en la base de datos `mycompany` de la siguiente forma:

1.3.1. Colección `departments`

Listing 1.3: Inserción en `departments`

```
use mycompany;

db.departments.insertMany([
  { _id: 10, dname: "ACCOUNTING", loc: "NEW YORK" },
  { _id: 20, dname: "RESEARCH",   loc: "DALLAS"   },
  { _id: 30, dname: "SALES",      loc: "CHICAGO"  }
]);
```

1.3.2. Colección `employees`

Listing 1.4: Inserción en `employees`

```
db.employees.insertMany([
  {
    _id: 7369,
    name: "SMITH",
    job: "CLERK",
    managerId: 7902,
    hiredate: new Date("1980-12-17"),
    sal: 800,
    comm: null,
    deptId: 20
  },
```

```
  {
    _id: 7499,
    name: "ALLEN",
    job: "SALESMAN",
    managerId: 7698,
    hiredate: new Date("1981-02-20"),
    sal: 1600,
    comm: 300,
    deptId: 30
  },
  {
    _id: 7521,
    name: "WARD",
    job: "SALESMAN",
    managerId: 7698,
    hiredate: new Date("1981-02-22"),
    sal: 1250,
    comm: 500,
    deptId: 30
  },
  {
    _id: 7566,
    name: "JONES",
    job: "MANAGER",
    managerId: 7839,
    hiredate: new Date("1981-04-02"),
    sal: 2975,
    comm: null,
    deptId: 20
  },
  {
    _id: 7698,
    name: "BLAKE",
    job: "MANAGER",
    managerId: 7839,
    hiredate: new Date("1981-05-01"),
    sal: 2850,
    comm: null,
    deptId: 30
  },
  {
    _id: 7782,
    name: "CLARK",
    job: "MANAGER",
    managerId: 7839,
    hiredate: new Date("1981-06-09"),
    sal: 2450,
    comm: null,
    deptId: 10
  },
  {
    _id: 7839,
    name: "KING",
    job: "PRESIDENT",
    // managerId: null o sin campo managerId
    hiredate: new Date("1981-11-17"),
    sal: 5000,
    comm: null,
    deptId: 10
  }
]);
```

1.4. Consultas de Validación

Con los datos cargados, se pueden realizar diversas consultas para verificar el acceso a la información:

1.4.1. Listar todos los empleados

Listing 1.5: Listado de empleados

```
db.employees.find().pretty();

[
  {
    _id: 7369,
    name: 'SMITH',
    job: 'CLERK',
    managerId: 7902,
    hiredate: ISODate('1980-12-17T00:00:00.000Z'),
    sal: 800,
    comm: null,
    deptId: 20
  },
  {
    _id: 7499,
    name: 'ALLEN',
    job: 'SALESMAN',
    managerId: 7698,
    hiredate: ISODate('1981-02-20T00:00:00.000Z'),
    sal: 1600,
    comm: 300,
    deptId: 30
  },
  {
    _id: 7521,
    name: 'WARD',
    job: 'SALESMAN',
    managerId: 7698,
    hiredate: ISODate('1981-02-22T00:00:00.000Z'),
    sal: 1250,
    comm: 500,
    deptId: 30
  },
  {
    _id: 7566,
    name: 'JONES',
    job: 'MANAGER',
    managerId: 7839,
    hiredate: ISODate('1981-04-02T00:00:00.000Z'),
    sal: 2975,
    comm: null,
    deptId: 20
  },
  {
    _id: 7698,
    name: 'BLAKE',
    job: 'MANAGER',
    managerId: 7839,
    hiredate: ISODate('1981-05-01T00:00:00.000Z'),
    sal: 2850,
    comm: null,
```

```
    deptId: 30
  },
  {
    _id: 7782,
    name: 'CLARK',
    job: 'MANAGER',
    managerId: 7839,
    hiredate: ISODate('1981-06-09T00:00:00.000Z'),
    sal: 2450,
    comm: null,
    deptId: 10
  },
  {
    _id: 7839,
    name: 'KING',
    job: 'PRESIDENT',
    hiredate: ISODate('1981-11-17T00:00:00.000Z'),
    sal: 5000,
    comm: null,
    deptId: 10
  }
]
```

1.4.2. Empleados con salario mayor a 2000

Listing 1.6: Empleados con `sal` ¿2000

```
db.employees.find(
  { sal: { $gt: 2000 } },
  { _id: 1, name: 1, job: 1, sal: 1 }
);

[
  { _id: 7566, name: 'JONES', job: 'MANAGER', sal: 2975 },
  { _id: 7698, name: 'BLAKE', job: 'MANAGER', sal: 2850 },
  { _id: 7782, name: 'CLARK', job: 'MANAGER', sal: 2450 },
  { _id: 7839, name: 'KING', job: 'PRESIDENT', sal: 5000 }
]
```

1.4.3. Contar empleados por departamento

Listing 1.7: Total de empleados por `deptId`

```
db.employees.aggregate([
  { $group: { _id: "$deptId", totalEmpleados: { $sum: 1 } } }
]);

[
  { _id: 20, totalEmpleados: 2 },
  { _id: 30, totalEmpleados: 3 },
  { _id: 10, totalEmpleados: 2 }
]
```

1.4.4. Unir empleados con su departamento (`$lookup`)

Listing 1.8: Unión empleado-departamento

```
db.employees.aggregate([
  {
    $lookup: {
      from: "departments",
      localField: "deptId",
      foreignField: "_id",
      as: "deptInfo"
    }
  }
]);

[
  {
    _id: 7369,
    name: 'SMITH',
    job: 'CLERK',
    managerId: 7902,
    hiredate: ISODate('1980-12-17T00:00:00.000Z'),
    sal: 800,
    comm: null,
    deptId: 20,
    deptInfo: [ { _id: 20, dname: 'RESEARCH', loc: 'DALLAS' } ]
  },
  {
    _id: 7499,
    name: 'ALLEN',
    job: 'SALESMAN',
    managerId: 7698,
    hiredate: ISODate('1981-02-20T00:00:00.000Z'),
    sal: 1600,
    comm: 300,
    deptId: 30,
    deptInfo: [ { _id: 30, dname: 'SALES', loc: 'CHICAGO' } ]
  },
  {
    _id: 7521,
    name: 'WARD',
    job: 'SALESMAN',
    managerId: 7698,
    hiredate: ISODate('1981-02-22T00:00:00.000Z'),
    sal: 1250,
    comm: 500,
    deptId: 30,
    deptInfo: [ { _id: 30, dname: 'SALES', loc: 'CHICAGO' } ]
  },
  {
    _id: 7566,
    name: 'JONES',
    job: 'MANAGER',
    managerId: 7839,
    hiredate: ISODate('1981-04-02T00:00:00.000Z'),
    sal: 2975,
    comm: null,
    deptId: 20,
    deptInfo: [ { _id: 20, dname: 'RESEARCH', loc: 'DALLAS' } ]
  },
  {
    _id: 7698,
```

```
    name: 'BLAKE',
    job: 'MANAGER',
    managerId: 7839,
    hiredate: ISODate('1981-05-01T00:00:00.000Z'),
    sal: 2850,
    comm: null,
    deptId: 30,
    deptInfo: [ { _id: 30, dname: 'SALES', loc: 'CHICAGO' } ]
  },
  {
    _id: 7782,
    name: 'CLARK',
    job: 'MANAGER',
    managerId: 7839,
    hiredate: ISODate('1981-06-09T00:00:00.000Z'),
    sal: 2450,
    comm: null,
    deptId: 10,
    deptInfo: [ { _id: 10, dname: 'ACCOUNTING', loc: 'NEW YORK' } ]
  },
  {
    _id: 7839,
    name: 'KING',
    job: 'PRESIDENT',
    hiredate: ISODate('1981-11-17T00:00:00.000Z'),
    sal: 5000,
    comm: null,
    deptId: 10,
    deptInfo: [ { _id: 10, dname: 'ACCOUNTING', loc: 'NEW YORK' } ]
  }
]
```

1.4.5. Mostrar nombre del jefe de cada empleado

Listing 1.9: Relación *jefe-subordinado*

```
db.employees.aggregate([
  {
    $lookup: {
      from: "employees",
      localField: "managerId",
      foreignField: "_id",
      as: "managerInfo"
    }
  },
  {
    $project: {
      name: 1,
      job: 1,
      manager: { $arrayElemAt: ["$managerInfo.name", 0] }
    }
  }
]);

[
  { _id: 7369, name: 'SMITH', job: 'CLERK' },
  { _id: 7499, name: 'ALLEN', job: 'SALESMAN', manager: 'BLAKE' },
  { _id: 7521, name: 'WARD', job: 'SALESMAN', manager: 'BLAKE' },
  { _id: 7566, name: 'JONES', job: 'MANAGER', manager: 'KING' },
  { _id: 7698, name: 'BLAKE', job: 'MANAGER', manager: 'KING' },
  { _id: 7782, name: 'CLARK', job: 'MANAGER', manager: 'KING' },
  { _id: 7839, name: 'KING', job: 'PRESIDENT' }
]
```

Capítulo 2

Ejercicio 2

2.1. Objetivo

El objetivo de esta práctica es afianzar los conocimientos impartidos en la parte teórica sobre la caracterización y predicción de actividades o propiedades de compuestos. En concreto, se pretende realizar la predicción (clasificación) del tipo de actividad biológica (Activo “1”; Inactivo “0”) a partir del *fingerprint* molecular de cada compuesto.

2.2. Trabajo a Realizar

1. Conectar a las bases de datos `CDS16` y `CDS29` (u otra base de datos similar) que contienen las colecciones `mfp_counts` y `molecules`.

2. Extraer los datos de la colección `molecules`, donde el campo `class` indica la actividad biológica.

3. Reconstruir la matriz de variables predictoras (*fingerprint* de 1024 bits, en columnas `FP1`, `FP2`, ..., `FP1024`) a partir de las posiciones que están a “1”.

4. Eliminar columnas en las que todos los valores sean “0”.

5. Utilizar alguno de los modelos de `scikit-learn` (`RandomForestClassifier`, `SVC`, etc.) para entrenar y predecir la actividad.

6. Evaluar el desempeño con métricas como *Accuracy*, *Kappa*, *AUC*, *GMean*, etc.

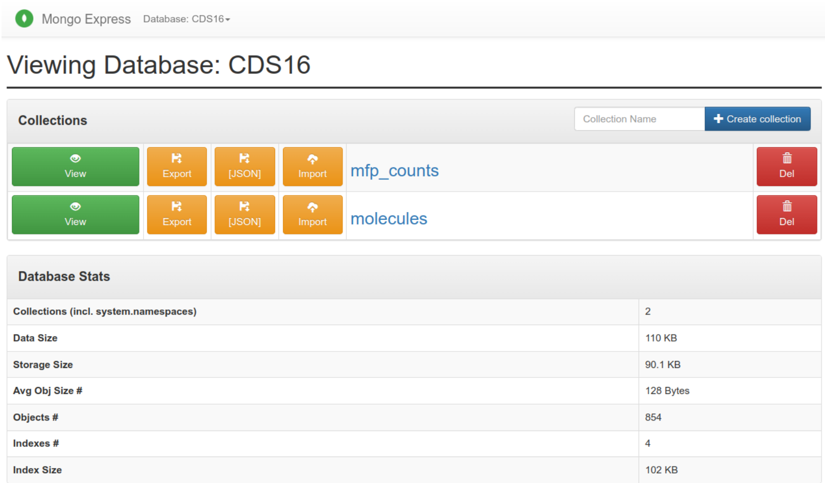

Figura 2.1: Base de Datos CDS16

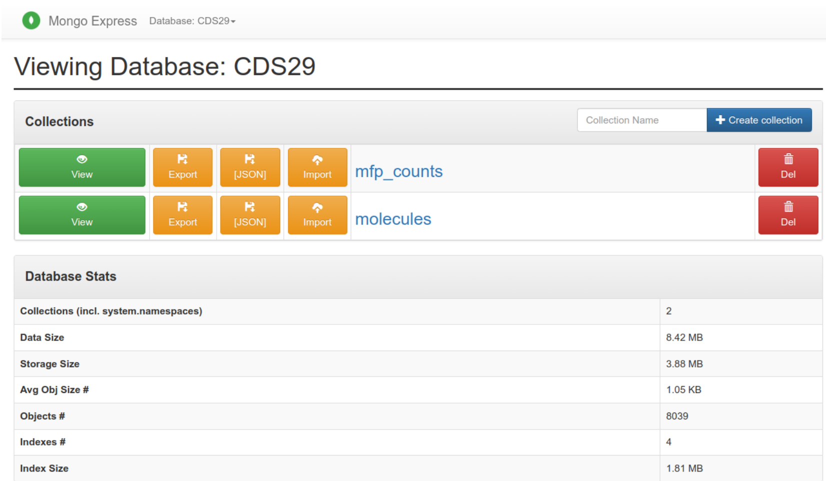

Figura 2.2: Base de Datos CDS29

2.3. Implementación en Python

Listing 2.1: Ejemplo unificado para la conexión y predicción

```python
import pymongo
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, cohen_kappa_score, roc_auc_score

# ------------------------------------------------------------------------------
# Helper function to load and process a dataset from a given collection.
# ------------------------------------------------------------------------------
def load_dataset(collection, num_bits=1024):
    docs = list(collection.find({}))
    df_rows = []

    for doc in docs:
        # Create a fingerprint array of 1024 bits initialized to 0.
        fp_row = np.zeros(num_bits, dtype=int)

        # Update the array if the document contains a valid fingerprint list.
        if "fingerprint" in doc and doc["fingerprint"]:
            for pos in doc["fingerprint"]:
                if 0 <= pos < num_bits:
                    fp_row[pos] = 1

        # Build a dictionary with "class" and each fingerprint bit.
        row_data = {"class": doc.get("class", 0)}  # default to 0 if "class" is missing
        for i in range(num_bits):
            row_data[f"FP{i+1}"] = fp_row[i]
        df_rows.append(row_data)

    # Convert list of dictionaries to a DataFrame.
    df = pd.DataFrame(df_rows)
    print("Before filtering, DataFrame shape:", df.shape)

    # Filter out fingerprint columns that are all zeros.
    filtered_df = df.loc[:, (df != 0).any(axis=0)]
    # If filtering removes all fingerprint columns (leaving only "class"), revert.
    if len(filtered_df.columns) <= 1:
        print("Warning: No non-zero fingerprint columns found; reverting to original DataFrame.")
        filtered_df = df

    print("After filtering, DataFrame shape:", filtered_df.shape)
    return filtered_df

```

```python
# -----------------------------------------------------------------------------
# Helper function to perform grid search for hyperparameter tuning.
# -----------------------------------------------------------------------------
def grid_search_rf(X_train, y_train):
    param_grid = {
        'n_estimators': [100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5],
    }
    rf = RandomForestClassifier(random_state=42, class_weight='balanced')
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    grid_search = GridSearchCV(rf, param_grid, cv=cv, scoring='roc_auc', n_jobs=-1)
    grid_search.fit(X_train, y_train)
    print("Best parameters:", grid_search.best_params_)
    return grid_search.best_estimator_

# -----------------------------------------------------------------------------
# Helper function to train and evaluate a model on a DataFrame.
# -----------------------------------------------------------------------------
def train_and_evaluate(df):
    # Separate features (X) and target (y)
    X = df.drop('class', axis=1)
    y = df['class']

    # Print class distribution to assess imbalance.
    print("Class distribution:")
    print(y.value_counts())

    # Split into training and testing sets (using stratification).
    X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.2, random_state=42, stratify=y
    )

    # Hyperparameter tuning with GridSearchCV.
    best_rf = grid_search_rf(X_train, y_train)

    # Train the tuned classifier on the full training set.
    best_rf.fit(X_train, y_train)

    # Predict on test data.
    y_pred = best_rf.predict(X_test)
    y_pred_prob = best_rf.predict_proba(X_test)[:, 1]

    # Calculate evaluation metrics.
    acc = accuracy_score(y_test, y_pred)
    kappa = cohen_kappa_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_pred_prob)

    # Plot histogram of predicted probabilities for diagnostics.
    plt.hist(y_pred_prob, bins=20)
    plt.xlabel("Predicted Probability")
    plt.ylabel("Frequency")
```

```python
    plt.title("Distribution of Predicted Probabilities")
    plt.show()

    return best_rf, acc, kappa, auc

# ------------------------------------------------------------------------------
# 1) Connect to MongoDB.
# ------------------------------------------------------------------------------
client = pymongo.MongoClient("mongodb://admin:1234@localhost:27017/")

# ------------------------------------------------------------------------------
# 2) Select the CDS16 database and its collections.
# ------------------------------------------------------------------------------
db_cds16 = client["CDS16"]
collection_molecules_16 = db_cds16["molecules"]
collection_mfp_counts_16 = db_cds16["mfp_counts"]

print(f"CDS16 - molecules: {collection_molecules_16.count_documents({})} documentos.")
print(f"CDS16 - mfp_counts: {collection_mfp_counts_16.count_documents({})} documentos.")

# ------------------------------------------------------------------------------
# 3) Select the CDS29 database and its collections.
# ------------------------------------------------------------------------------
db_cds29 = client["CDS29"]
collection_molecules_29 = db_cds29["molecules"]
collection_mfp_counts_29 = db_cds29["mfp_counts"]

print(f"CDS29 - molecules: {collection_molecules_29.count_documents({})} documentos.")
print(f"CDS29 - mfp_counts: {collection_mfp_counts_29.count_documents({})} documentos.")

# ------------------------------------------------------------------------------
# 4) Load and process datasets from both CDS16 and CDS29.
# ------------------------------------------------------------------------------
df_cds16 = load_dataset(collection_molecules_16)
df_cds29 = load_dataset(collection_molecules_29)

# Ensure that the feature matrix is not empty.
if df_cds16.drop('class', axis=1).empty:
    raise ValueError("Feature matrix for CDS16 is empty. Please check your data ingestion and filtering steps.")
if df_cds29.drop('class', axis=1).empty:
```

```
    raise ValueError("Feature matrix for CDS29 is empty. Please check your data ingestion and filtering steps.")

# ------------------------------------------------------------------------------
# 5) Train and evaluate the model on both datasets.
# ------------------------------------------------------------------------------
rf_cds16, acc_cds16, kappa_cds16, auc_cds16 = train_and_evaluate(df_cds16)
rf_cds29, acc_cds29, kappa_cds29, auc_cds29 = train_and_evaluate(df_cds29)

# ------------------------------------------------------------------------------
# 6) Print and compare performance metrics.
# ------------------------------------------------------------------------------
comparison_df = pd.DataFrame({
    "Dataset": ["CDS16", "CDS29"],
    "Accuracy": [acc_cds16, acc_cds29],
    "Kappa": [kappa_cds16, kappa_cds29],
    "AUC": [auc_cds16, auc_cds29]
})

print("\nComparison of model performance:")
print(comparison_df)

Comparison of model performance:
  Dataset  Accuracy  Kappa  AUC
0   CDS16  0.470588    0.0  0.5
1   CDS29  0.803991    0.0  0.5
```