

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)



UNIVERSIDAD
DE CÓRDOBA

Autor:

Marcos Rivera Gavilán

Profesor:

Gonzalo Cerrueda García

Índice general

Índice general	1
1. Ejercicio 1	2
1.1. Objetivo	2
1.2. Propuesta de Estructura de Colecciones	2
1.3. Carga de Información de Prueba	4
1.3.1. Colección departments	4
1.3.2. Colección employees	4
1.4. Consultas de Validación	6
1.4.1. Listar todos los empleados	6
1.4.2. Empleados con salario mayor a 2000	7
1.4.3. Contar empleados por departamento	7
1.4.4. Unir empleados con su departamento (\$lookup)	8
1.4.5. Mostrar nombre del jefe de cada empleado	10
2. Ejercicio 2	11
2.1. Objetivo	11
2.2. Trabajo a Realizar	11
2.3. Implementación en Python	13

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).

Mongo Express Database: mycompany -> Collection: departments -

Viewing Collection: departments

New Document New Index

Simple Advanced

Key Value String Find

Delete all 3 documents retrieved

_id	dname	loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO

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

Mongo Express Database: mycompany -> Collection: employees -

Viewing Collection: employees

New Document New Index

Simple Advanced

Key Value String Find

Delete all 7 documents retrieved

_id	name	job	managerid	hiredate	sal	comm	deptid
7369	SMITH	CLERK	7902	Wed Dec 17 1980 00:00:00 GMT+0000 (Coordinated Universal Time)	800	20	
7499	ALLEN	SALESMAN	7698	Fri Feb 20 1981 00:00:00 GMT+0000 (Coordinated Universal Time)	1600	300	30
7521	WARD	SALESMAN	7698	Sun Feb 22 1981 00:00:00 GMT+0000 (Coordinated Universal Time)	1250	500	30
7566	JONES	MANAGER	7839	Thu Apr 02 1981 00:00:00 GMT+0000 (Coordinated Universal Time)	2975	20	
7698	BLAKE	MANAGER	7839	Fri May 01 1981 00:00:00 GMT+0000 (Coordinated Universal Time)	2850	30	
7782	CLARK	MANAGER	7839	Tue Jun 09 1981 00:00:00 GMT+0000 (Coordinated Universal Time)	2450	10	
7839	KING	PRESIDENT		Tue Nov 17 1981 00:00:00 GMT+0000 (Coordinated Universal Time)	5000	10	

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`

```
1 {
2   "_id": 10,
3   "dname": "ACCOUNTING",
4   "loc": "NEW YORK"
5 }
```

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

Listing 1.2: Ejemplo de documento en `employees`

```
1 {
2   "_id": 7369,
3   "name": "SMITH",
4   "job": "CLERK",
5   "managerId": 7902,
6   "hiredate": ISODate("1980-12-17"),
7   "sal": 800,
8   "comm": null,
9   "deptId": 20
10 }
```

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`

```
1 use mycompany;
2
3 db.departments.insertMany([
4   { _id: 10, dname: "ACCOUNTING", loc: "NEW YORK" },
5   { _id: 20, dname: "RESEARCH", loc: "DALLAS" },
6   { _id: 30, dname: "SALES", loc: "CHICAGO" }
7 ]);
```

1.3.2. Colección `employees`

Listing 1.4: Inserción en `employees`

```
1 db.employees.insertMany([
2   {
3     _id: 7369,
4     name: "SMITH",
5     job: "CLERK",
6     managerId: 7902,
7     hiredate: new Date("1980-12-17"),
8     sal: 800,
9     comm: null,
10    deptId: 20
11  },
```

```

12     {
13         _id: 7499,
14         name: "ALLEN",
15         job: "SALESMAN",
16         managerId: 7698,
17         hiredate: new Date("1981-02-20"),
18         sal: 1600,
19         comm: 300,
20         deptId: 30
21     },
22     {
23         _id: 7521,
24         name: "WARD",
25         job: "SALESMAN",
26         managerId: 7698,
27         hiredate: new Date("1981-02-22"),
28         sal: 1250,
29         comm: 500,
30         deptId: 30
31     },
32     {
33         _id: 7566,
34         name: "JONES",
35         job: "MANAGER",
36         managerId: 7839,
37         hiredate: new Date("1981-04-02"),
38         sal: 2975,
39         comm: null,
40         deptId: 20
41     },
42     {
43         _id: 7698,
44         name: "BLAKE",
45         job: "MANAGER",
46         managerId: 7839,
47         hiredate: new Date("1981-05-01"),
48         sal: 2850,
49         comm: null,
50         deptId: 30
51     },
52     {
53         _id: 7782,
54         name: "CLARK",
55         job: "MANAGER",
56         managerId: 7839,
57         hiredate: new Date("1981-06-09"),
58         sal: 2450,
59         comm: null,
60         deptId: 10
61     },
62     {
63         _id: 7839,
64         name: "KING",
65         job: "PRESIDENT",
66         // managerId: null o sin campo managerId
67         hiredate: new Date("1981-11-17"),
68         sal: 5000,
69         comm: null,
70         deptId: 10
71     }
72 ];

```

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

```
1 db.employees.find().pretty();
2
3 [
4   {
5     _id: 7369,
6     name: 'SMITH',
7     job: 'CLERK',
8     managerId: 7902,
9     hiredate: ISODate('1980-12-17T00:00:00.000Z'),
10    sal: 800,
11    comm: null,
12    deptId: 20
13  },
14  {
15    _id: 7499,
16    name: 'ALLEN',
17    job: 'SALESMAN',
18    managerId: 7698,
19    hiredate: ISODate('1981-02-20T00:00:00.000Z'),
20    sal: 1600,
21    comm: 300,
22    deptId: 30
23  },
24  {
25    _id: 7521,
26    name: 'WARD',
27    job: 'SALESMAN',
28    managerId: 7698,
29    hiredate: ISODate('1981-02-22T00:00:00.000Z'),
30    sal: 1250,
31    comm: 500,
32    deptId: 30
33  },
34  {
35    _id: 7566,
36    name: 'JONES',
37    job: 'MANAGER',
38    managerId: 7839,
39    hiredate: ISODate('1981-04-02T00:00:00.000Z'),
40    sal: 2975,
41    comm: null,
42    deptId: 20
43  },
44  {
45    _id: 7698,
46    name: 'BLAKE',
47    job: 'MANAGER',
48    managerId: 7839,
49    hiredate: ISODate('1981-05-01T00:00:00.000Z'),
50    sal: 2850,
51    comm: null,
```

```

52     deptId: 30
53   },
54   {
55     _id: 7782,
56     name: 'CLARK',
57     job: 'MANAGER',
58     managerId: 7839,
59     hiredate: ISODate('1981-06-09T00:00:00.000Z'),
60     sal: 2450,
61     comm: null,
62     deptId: 10
63   },
64   {
65     _id: 7839,
66     name: 'KING',
67     job: 'PRESIDENT',
68     hiredate: ISODate('1981-11-17T00:00:00.000Z'),
69     sal: 5000,
70     comm: null,
71     deptId: 10
72   }
73 ]

```

1.4.2. Empleados con salario mayor a 2000

Listing 1.6: Empleados con sal >2000

```

1 db.employees.find(
2   { sal: { $gt: 2000 } },
3   { _id: 1, name: 1, job: 1, sal: 1 }
4 );
5
6 [
7   { _id: 7566, name: 'JONES', job: 'MANAGER', sal: 2975 },
8   { _id: 7698, name: 'BLAKE', job: 'MANAGER', sal: 2850 },
9   { _id: 7782, name: 'CLARK', job: 'MANAGER', sal: 2450 },
10  { _id: 7839, name: 'KING', job: 'PRESIDENT', sal: 5000 }
11 ]

```

1.4.3. Contar empleados por departamento

Listing 1.7: Total de empleados por deptId

```

1 db.employees.aggregate([
2   { $group: { _id: "$deptId", totalEmpleados: { $sum: 1 } } }
3 ]);
4
5 [
6   { _id: 20, totalEmpleados: 2 },
7   { _id: 30, totalEmpleados: 3 },
8   { _id: 10, totalEmpleados: 2 }
9 ]

```

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

Listing 1.8: Unión empleado-departamento

```
1 db.employees.aggregate([
2   {
3     $lookup: {
4       from: "departments",
5       localField: "deptId",
6       foreignField: "_id",
7       as: "deptInfo"
8     }
9   }
10  ]);
11
12  [
13    {
14      _id: 7369,
15      name: 'SMITH',
16      job: 'CLERK',
17      managerId: 7902,
18      hiredate: ISODate('1980-12-17T00:00:00.000Z'),
19      sal: 800,
20      comm: null,
21      deptId: 20,
22      deptInfo: [ { _id: 20, dname: 'RESEARCH', loc: 'DALLAS' } ]
23    },
24    {
25      _id: 7499,
26      name: 'ALLEN',
27      job: 'SALESMAN',
28      managerId: 7698,
29      hiredate: ISODate('1981-02-20T00:00:00.000Z'),
30      sal: 1600,
31      comm: 300,
32      deptId: 30,
33      deptInfo: [ { _id: 30, dname: 'SALES', loc: 'CHICAGO' } ]
34    },
35    {
36      _id: 7521,
37      name: 'WARD',
38      job: 'SALESMAN',
39      managerId: 7698,
40      hiredate: ISODate('1981-02-22T00:00:00.000Z'),
41      sal: 1250,
42      comm: 500,
43      deptId: 30,
44      deptInfo: [ { _id: 30, dname: 'SALES', loc: 'CHICAGO' } ]
45    },
46    {
47      _id: 7566,
48      name: 'JONES',
49      job: 'MANAGER',
50      managerId: 7839,
51      hiredate: ISODate('1981-04-02T00:00:00.000Z'),
52      sal: 2975,
53      comm: null,
54      deptId: 20,
55      deptInfo: [ { _id: 20, dname: 'RESEARCH', loc: 'DALLAS' } ]
56    },
57    {
58      _id: 7698,
```

```

59     name: 'BLAKE',
60     job: 'MANAGER',
61     managerId: 7839,
62     hiredate: ISODate('1981-05-01T00:00:00.000Z'),
63     sal: 2850,
64     comm: null,
65     deptId: 30,
66     deptInfo: [ { _id: 30, dname: 'SALES', loc: 'CHICAGO' } ]
67 },
68 {
69     _id: 7782,
70     name: 'CLARK',
71     job: 'MANAGER',
72     managerId: 7839,
73     hiredate: ISODate('1981-06-09T00:00:00.000Z'),
74     sal: 2450,
75     comm: null,
76     deptId: 10,
77     deptInfo: [ { _id: 10, dname: 'ACCOUNTING', loc: 'NEW YORK' } ]
78 },
79 {
80     _id: 7839,
81     name: 'KING',
82     job: 'PRESIDENT',
83     hiredate: ISODate('1981-11-17T00:00:00.000Z'),
84     sal: 5000,
85     comm: null,
86     deptId: 10,
87     deptInfo: [ { _id: 10, dname: 'ACCOUNTING', loc: 'NEW YORK' } ]
88 }
89 ]

```

1.4.5. Mostrar nombre del jefe de cada empleado

Listing 1.9: Relación *jefe-subordinado*

```
1 db.employees.aggregate([
2   {
3     $lookup: {
4       from: "employees",
5       localField: "managerId",
6       foreignField: "_id",
7       as: "managerInfo"
8     }
9   },
10  {
11    $project: {
12      name: 1,
13      job: 1,
14      manager: { $arrayElemAt: ["$managerInfo.name", 0] }
15    }
16  }
17 ]);
18
19 [
20   { _id: 7369, name: 'SMITH', job: 'CLERK' },
21   { _id: 7499, name: 'ALLEN', job: 'SALESMAN', manager: 'BLAKE' },
22   { _id: 7521, name: 'WARD', job: 'SALESMAN', manager: 'BLAKE' },
23   { _id: 7566, name: 'JONES', job: 'MANAGER', manager: 'KING' },
24   { _id: 7698, name: 'BLAKE', job: 'MANAGER', manager: 'KING' },
25   { _id: 7782, name: 'CLARK', job: 'MANAGER', manager: 'KING' },
26   { _id: 7839, name: 'KING', job: 'PRESIDENT' }
27 ]
```

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.

Mongo Express Database: CDS16

Viewing Database: CDS16

Collections

Collection Name [+ Create collection](#)

View	Export	[JSON]	Import	mfp_counts	Del
View	Export	[JSON]	Import	molecules	Del

Database Stats

Collections (incl. system.namespaces)	2
Data Size	110 KB
Storage Size	90.1 KB
Avg Obj Size #	128 Bytes
Objects #	854
Indexes #	4
Index Size	102 KB

Figura 2.1: Base de Datos CDS16

Mongo Express Database: CDS29

Viewing Database: CDS29

Collections

Collection Name [+ Create collection](#)

View	Export	[JSON]	Import	mfp_counts	Del
View	Export	[JSON]	Import	molecules	Del

Database Stats

Collections (incl. system.namespaces)	2
Data Size	8.42 MB
Storage Size	3.88 MB
Avg Obj Size #	1.05 KB
Objects #	8039
Indexes #	4
Index Size	1.81 MB

Figura 2.2: Base de Datos CDS29

2.3. Implementación en Python

Listing 2.1: Ejemplo unificado para la conexión y predicción

```
1 import pymongo
2 import numpy as np
3 import pandas as pd
4 import matplotlib.pyplot as plt
5
6 from sklearn.model_selection import train_test_split, GridSearchCV,
   StratifiedKFold
7 from sklearn.ensemble import RandomForestClassifier
8 from sklearn.metrics import accuracy_score, cohen_kappa_score,
   roc_auc_score
9
10 # -----
11 # Helper function to load and process a dataset from a given
   collection.
12 # -----
13
14 def load_dataset(collection, num_bits=1024):
15     docs = list(collection.find({}))
16     df_rows = []
17
18     for doc in docs:
19         # Create a fingerprint array of 1024 bits initialized to 0.
20         fp_row = np.zeros(num_bits, dtype=int)
21
22         # Update the array if the document contains a valid
23         # fingerprint list.
24         if "fingerprint" in doc and doc["fingerprint"]:
25             for pos in doc["fingerprint"]:
26                 if 0 <= pos < num_bits:
27                     fp_row[pos] = 1
28
29         # Build a dictionary with "class" and each fingerprint bit.
30         row_data = {"class": doc.get("class", 0)} # default to 0
31         if "class" is missing
32         for i in range(num_bits):
33             row_data[f"FP{i+1}"] = fp_row[i]
34         df_rows.append(row_data)
35
36     # Convert list of dictionaries to a DataFrame.
37     df = pd.DataFrame(df_rows)
38     print("Before filtering, DataFrame shape:", df.shape)
39
40     # Filter out fingerprint columns that are all zeros.
41     filtered_df = df.loc[:, (df != 0).any(axis=0)]
42     # If filtering removes all fingerprint columns (leaving only "
43     # class"), revert.
44     if len(filtered_df.columns) <= 1:
45         print("Warning: No non-zero fingerprint columns found;
46             reverting to original DataFrame.")
47         filtered_df = df
48
49     print("After filtering, DataFrame shape:", filtered_df.shape)
50     return filtered_df
```

```

47 # -----

48 # Helper function to perform grid search for hyperparameter tuning.
49 # -----

50 def grid_search_rf(X_train, y_train):
51     param_grid = {
52         'n_estimators': [100, 200],
53         'max_depth': [None, 10, 20],
54         'min_samples_split': [2, 5],
55     }
56     rf = RandomForestClassifier(random_state=42, class_weight='
        balanced')
57     cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
58     grid_search = GridSearchCV(rf, param_grid, cv=cv, scoring='
        roc_auc', n_jobs=-1)
59     grid_search.fit(X_train, y_train)
60     print("Best parameters:", grid_search.best_params_)
61     return grid_search.best_estimator_
62
63 # -----

64 # Helper function to train and evaluate a model on a DataFrame.
65 # -----

66 def train_and_evaluate(df):
67     # Separate features (X) and target (y)
68     X = df.drop('class', axis=1)
69     y = df['class']
70
71     # Print class distribution to assess imbalance.
72     print("Class distribution:")
73     print(y.value_counts())
74
75     # Split into training and testing sets (using stratification).
76     X_train, X_test, y_train, y_test = train_test_split(
77         X, y, test_size=0.2, random_state=42, stratify=y
78     )
79
80     # Hyperparameter tuning with GridSearchCV.
81     best_rf = grid_search_rf(X_train, y_train)
82
83     # Train the tuned classifier on the full training set.
84     best_rf.fit(X_train, y_train)
85
86     # Predict on test data.
87     y_pred = best_rf.predict(X_test)
88     y_pred_prob = best_rf.predict_proba(X_test)[:, 1]
89
90     # Calculate evaluation metrics.
91     acc = accuracy_score(y_test, y_pred)
92     kappa = cohen_kappa_score(y_test, y_pred)
93     auc = roc_auc_score(y_test, y_pred_prob)
94
95     # Plot histogram of predicted probabilities for diagnostics.
96     plt.hist(y_pred_prob, bins=20)
97     plt.xlabel("Predicted Probability")
98     plt.ylabel("Frequency")

```

```

99     plt.title("Distribution of Predicted Probabilities")
100     plt.show()
101
102     return best_rf, acc, kappa, auc
103
104 # -----
105 # 1) Connect to MongoDB.
106 # -----
107 client = pymongo.MongoClient("mongodb://admin:1234@localhost:27017/"
108                               ")
109 # -----
110 # 2) Select the CDS16 database and its collections.
111 # -----
112 db_cds16 = client["CDS16"]
113 collection_molecules_16 = db_cds16["molecules"]
114 collection_mfp_counts_16 = db_cds16["mfp_counts"]
115
116 print(f"CDS16 - molecules: {collection_molecules_16.count_documents(
117       {}}) documents.")
117 print(f"CDS16 - mfp_counts: {collection_mfp_counts_16.
118       count_documents({})} documents.")
118
119 # -----
120 # 3) Select the CDS29 database and its collections.
121 # -----
122 db_cds29 = client["CDS29"]
123 collection_molecules_29 = db_cds29["molecules"]
124 collection_mfp_counts_29 = db_cds29["mfp_counts"]
125
126 print(f"CDS29 - molecules: {collection_molecules_29.count_documents(
127       {}}) documents.")
127 print(f"CDS29 - mfp_counts: {collection_mfp_counts_29.
128       count_documents({})} documents.")
128
129 # -----
130 # 4) Load and process datasets from both CDS16 and CDS29.
131 # -----
132 df_cds16 = load_dataset(collection_molecules_16)
133 df_cds29 = load_dataset(collection_molecules_29)
134
135 # Ensure that the feature matrix is not empty.
136 if df_cds16.drop('class', axis=1).empty:
137     raise ValueError("Feature matrix for CDS16 is empty. Please
138                       check your data ingestion and filtering steps.")
138 if df_cds29.drop('class', axis=1).empty:

```



```

139         raise ValueError("Feature matrix for CDS29 is empty. Please
140                             check your data ingestion and filtering steps.")
141     # -----
142     # 5) Train and evaluate the model on both datasets.
143     # -----
144     rf_cds16, acc_cds16, kappa_cds16, auc_cds16 = train_and_evaluate(
145         df_cds16)
146     rf_cds29, acc_cds29, kappa_cds29, auc_cds29 = train_and_evaluate(
147         df_cds29)
148     # -----
149     # 6) Print and compare performance metrics.
150     # -----
151     comparison_df = pd.DataFrame({
152         "Dataset": ["CDS16", "CDS29"],
153         "Accuracy": [acc_cds16, acc_cds29],
154         "Kappa": [kappa_cds16, kappa_cds29],
155         "AUC": [auc_cds16, auc_cds29]
156     })
157     print("\nComparison of model performance:")
158     print(comparison_df)
159
160     Comparison of model performance:
161     Dataset Accuracy Kappa AUC
162     0    CDS16  0.470588   0.0  0.5
163     1    CDS29  0.803991   0.0  0.5

```
