

Modulo 1: Bayesian Classifier

PUCP — NoSQL — 2025

Introducción

La detección de fraude en sistemas financieros es una tarea crítica.

Este proyecto presenta un enfoque probabilístico para inferir si una transacción fue fraudulenta, usando técnicas de inferencia bayesiana y estructuras causales definidas por el usuario.

https://github.com/daneelsan/PUCP_NoSQL_BayesianClassifier/tree/main/BayesianClassifier

The interface has two tabs: 'Inferencia' (active) and 'Hipótesis'. Under 'Inferencia', there are instructions in Spanish. Below the instructions, there are five dropdown menus for 'Hypothesis', 'Gender', 'Age', 'Category', and 'Amount'. The selected values are: Naive Bayes, F, 2, es_travel, and medium. At the bottom, there is a green '✓ Clasificar' button and a green checkmark icon.

(a) Visualización de hipótesis 1

The interface is similar to the first one, but the 'Amount' dropdown is set to 'very low'. At the bottom, there is a green '✓ Clasificar' button and a red '✗ Invalid' message.

(b) Visualización de hipótesis 2

Figura 1: Comparación de estructuras de hipótesis

fraud_credit_card.csv

La data representa transacciones financieras.

Cada fila es una transacción individual y contiene información detallada sobre ella:

- **step**: Un paso o índice de tiempo.
- **customer**: Un identificador único para el cliente que realiza la transacción.
- **age**: La edad del cliente, probablemente en categorías o rangos ('0', '1', '2', etc., 'U' para desconocido).
- **gender**: El género del cliente ('M' para masculino, 'F' para femenino, y posiblemente 'E' o 'U' para otros/desconocido).
- **zipcodeOri**: El código postal de origen de la transacción.
- **merchant**: Un identificador único para el comercio donde se realizó la transacción.
- **zipMerchant**: El código postal del comercio.
- **category**: La categoría o tipo de la transacción (ej. 'es_transportation' para transporte, 'es_health' para salud).
- **amount**: El valor monetario de la transacción.
 - Importante: Utiliza coma (,) como separador decimal (ej. "4,55" es 4.55).
- **fraud**: La variable objetivo. Es un indicador binario (0 o 1) que señala si la transacción fue fraudulenta (1) o no (0).

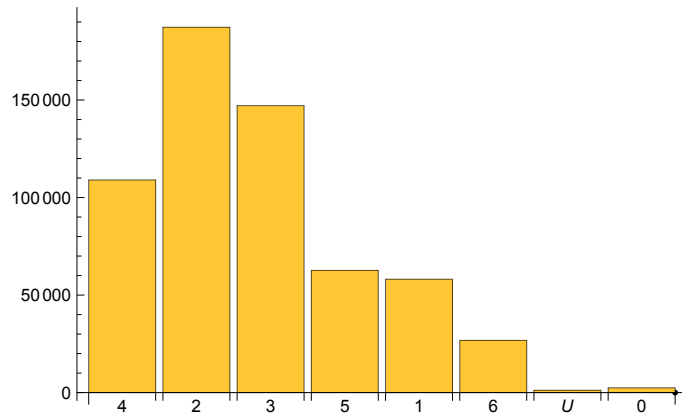
En resumen, son registros de transacciones con características del cliente, del comercio y de la propia operación, etiquetadas como **fraudulentas** o **no fraudulentas**.

Out[8]=

	step	customer	age	gender	zipcodeOri	merchant	zipMerchant	
1	0	'C1093826151'	4	M	'28007'	'M348934600'	'28007'	
2	0	'C352968107'	2	M	'28007'	'M348934600'	'28007'	
3	0	'C2054744914'	4	F	'28007'	'M1823072687'	'28007'	
4	0	'C1760612790'	3	M	'28007'	'M348934600'	'28007'	
5	0	'C757503768'	5	M	'28007'	'M348934600'	'28007'	
6	0	'C1315400589'	3	F	'28007'	'M348934600'	'28007'	

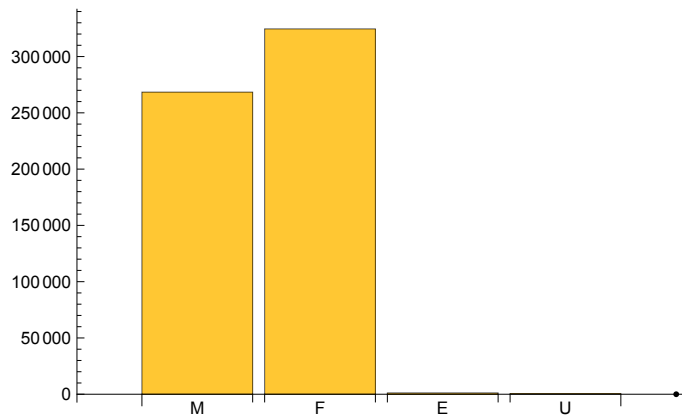
age

```
In[ ]:= BarChart[Counts[original[All, "age"]], ChartLabels → Automatic]  
Out[ ]=
```



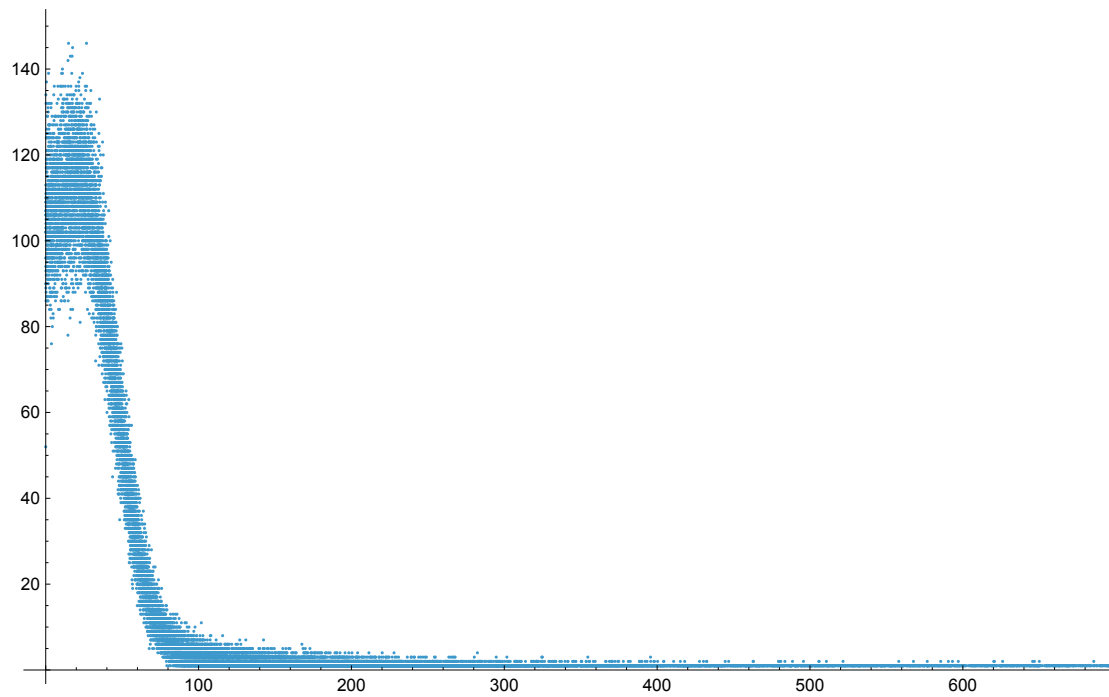
gender

```
In[ ]:= BarChart[Counts[original[All, "gender"]], ChartLabels → Automatic]  
Out[ ]=
```



amount

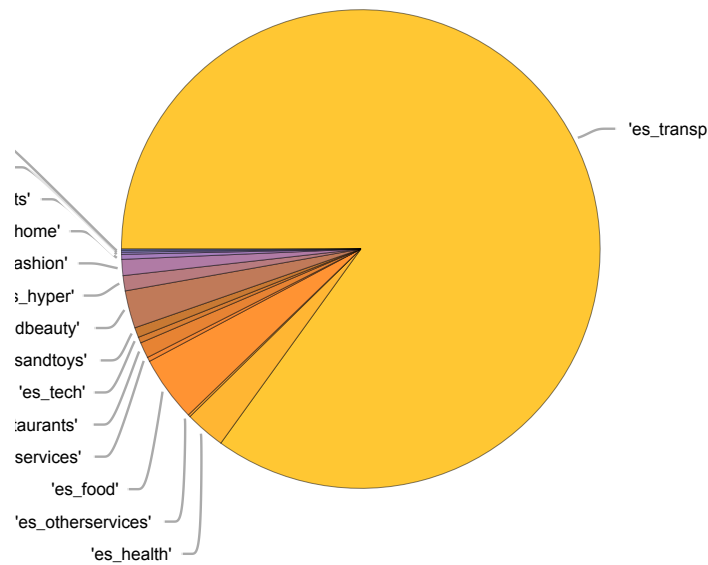
```
In[ ]:= ListPlot[Counts[original[[All, "amount"]]], ImageSize -> Large]
Out[ ]:=
```

**category**

```
In[11]:= ReverseSort[Dataset[Counts[original[[All, "category"]]]]]
Out[11]=
```

'es_transportation'	505119
'es_food'	26254
'es_health'	16133
'es_wellnessandbeauty'	15086
'es_fashion'	6454
'es_barsandrestaurants'	6373
'es_hyper'	6098
'es_sportsandtoys'	4002
'es_tech'	2370
'es_home'	1986
'es_hotelservices'	1744
'es_otherservices'	912
'es_contents'	885
'es_travel'	728
'es_leisure'	499

```
In[10]:= PieChart[Counts[original[All, "category"]],  
  ChartLabels -> Callout[Automatic], ImageSize -> Medium]  
Out[10]=
```



fraud

Muchos más “no fraude” que “fraudes”:

```
In[* ]:= Counts[original[All, "fraud"]]  
Out[* ]:=  
<| 0 -> 587 443, 1 -> 7200 |>  
Out[* ]//TableForm=
```

	count	percentage
no fraud	587 443	98.7892%
fraud	7200	1.21081%

upload_dataset.py

Objetivo: **preparar y cargar datos de transacciones financieras en una base de datos MongoDB:**

1. Carga de Datos:

- Lee un archivo CSV llamado `fraud_credit_card.csv` en un DataFrame de Pandas.

2. Preprocesamiento y Limpieza de Datos:

- Convierte el campo `amount` (monto) a formato numérico (reemplazando comas por puntos).
- Discretiza el `amount` en categorías como 'very low', 'low', 'medium' y 'high'.
- Estandariza los campos `age`, `gender` y `category` (remueve comillas y maneja valores 'U' o desconocidos).
- Convierte la columna `fraud` (que es 0 o 1) a etiquetas 'yes' o 'no'.

3. Selección de Características:

- Elimina columnas que se consideran no relevantes para el análisis posterior (como `step`, `customer`, `zipcodeOri`, `merchant`, `zipMerchant`).

4. Carga a MongoDB:

- Establece una conexión a una base de datos MongoDB Atlas (usando credenciales de variables de entorno).
- Limpia (borra) la colección `transactions` en la base de datos `fraud_db`.
- Inserta los datos preprocesados del DataFrame en la colección `transactions` de MongoDB, realizando la carga en **batches** para optimizar el rendimiento y mostrando una barra de progreso.

index_dataset.py

Objetivo: **preparar, indexar y optimizar un dataset de transacciones almacenado en MongoDB** para su uso *eficiente* en un clasificador bayesiano.

1. Calcular y Almacenar Cardinalidades:

- Identifica todos los valores únicos para las variables clave y les asigna un índice numérico.
- Estas correspondencias (mapeos) se guardan en la colección `cardinalities`.

2. Indexar el Dataset:

- Crea la colección `transactions_indexed` donde los valores categóricos originales de las transacciones se reemplazan por sus índices numéricos correspondientes.

3. Precalcular y Almacenar Conteos:

- Calcula y guarda en la colección `precomputed` las frecuencias de aparición de valores individuales y ciertas combinaciones de valores (especialmente con la variable `fraud`).
- Actúa como una caché para acelerar las consultas de probabilidad realizadas por el clasificador.

MongoDB

transactions

La data del archivo **fraud_credit_card.xlsx**:

```
self.client["fraud_db"]["transactions"]
```

ClusterPUCP > fraud_db > transactions Open MongoDB shell

Documents 594.6K Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#) Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE 25 1 - 25 of 594643 ↺ ↻ ↷ ⌵ ⌶ ⌷

<pre>{ "_id": ObjectId('6854e243e35a50901882de2f'), "age": "5", "gender": "F", "category": "es_transportation", "amount_bin": "very low", "fraud": "no" }</pre>
<pre>{ "_id": ObjectId('6854e243e35a50901882de32'), "age": "4", "gender": "F", "category": "es_health", "amount_bin": "high", "fraud": "no" }</pre>
<pre>{ "_id": ObjectId('6854e243e35a50901882de37'), "age": "3", "gender": "F", "category": "es_transportation", "amount_bin": "low", "fraud": "no" }</pre>
<pre>{ "_id": ObjectId('6854e243e35a50901882de57'), "age": "2", "gender": "M" }</pre>

MongoDB

cardinalities

Guarda las cardinalidades de los parámetros:

```
self.client["fraud_db"]["cardinalities"]
```

ClusterPUCP > fraud_db > cardinalities Open MongoDB shell

Documents 5 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#) Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE 25 1 - 5 of 5

- `_id: ObjectId('6857582f5b0e094b81cdd54b')`
`variable: "age"`
`mapping: Object`
- `_id: ObjectId('6857582f5b0e094b81cdd54c')`
`variable: "gender"`
`mapping: Object`
 - `M: 0`
 - `F: 1`
 - `E: 2`
 - `U: 3`
- `_id: ObjectId('6857582f5b0e094b81cdd54d')`
`variable: "category"`
`mapping: Object`
- `_id: ObjectId('6857582f5b0e094b81cdd54e')`
`variable: "amount_bin"`
`mapping: Object`
- `_id: ObjectId('6857582f5b0e094b81cdd54f')`
`variable: "fraud"`
`mapping: Object`

MongoDB

transactions_indexed

Versión indexada del collection *transactions*:

```
self.client["fraud_db"]["transactions_indexed"]
```

ClusterPUCP > fraud_db > transactions_indexed Open MongoDB shell

Documents 594.6K Aggregations Schema Indexes 5 Validation

Type a query: { field: 'value' } or [Generate query](#) Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE 25 1 - 25 of 594643

```
{
  "_id": ObjectId('6854e243e35a50901882de19'),
  "age": 3,
  "gender": 1,
  "category": 0,
  "amount_bin": 1,
  "fraud": 0
}
```

```
{
  "_id": ObjectId('6854e243e35a50901882de1f'),
  "age": 3,
  "gender": 1,
  "category": 0,
  "amount_bin": 1,
  "fraud": 0
}
```

```
{
  "_id": ObjectId('6854e243e35a50901882de49'),
  "age": 5,
  "gender": 0,
  "category": 0,
  "amount_bin": 1,
  "fraud": 0
}
```

```
{
  "_id": ObjectId('6854e243e35a50901882de55'),
  "age": 1,
  "gender": 1
}
```

MongoDB

precomputed

Conteos precomputados para optimizar la clasificación:

```
self.client["fraud_db"]["precomputed"]
```

ClusterPUCP > fraud_db > precomputed Open MongoDB shell

Documents (101) Aggregations Schema Indexes (1) Validation

Type a query: { field: 'value' } or [Generate query](#) Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE 25 1 - 25 of 101 ↺ ↻ ↷ ⌵ ⌶ ⌷

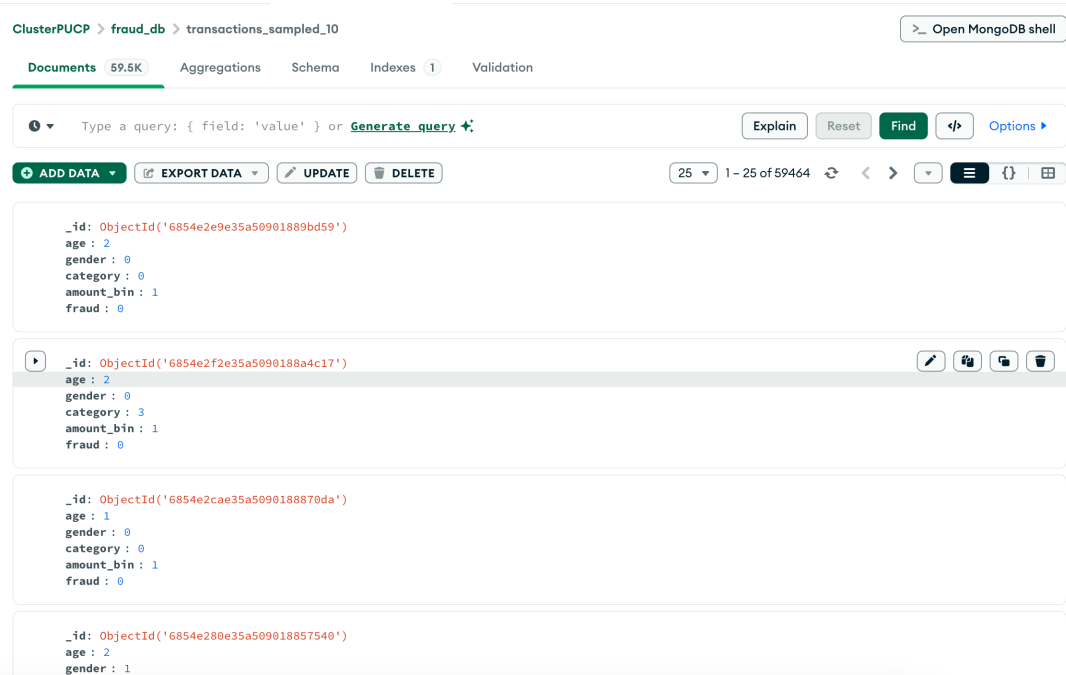
<pre>{ "_id": ObjectId("6857598c5b0e094b81cdd550"), "fraud": 0, "count": 587443 }</pre>
<pre>{ "_id": ObjectId("6857598d5b0e094b81cdd551"), "fraud": 1, "count": 7200 }</pre>
<pre>{ "_id": ObjectId("6857598d5b0e094b81cdd552"), "age": 0, "count": 109025 }</pre>
<pre>{ "_id": ObjectId("6857598d5b0e094b81cdd553"), "age": 0, "fraud": 0, "count": 107615 }</pre>
<pre>{ "_id": ObjectId("6857598e5b0e094b81cdd554"), "age": 0, "fraud": 1, "count": 1410 }</pre>

MongoDB

transactions_sampled_10

Un sample del 10% de la colección “transacions_indexed”:

```
self.client["fraud_db"]["transactions_sampled_10"]
```



ClusterPUCP > fraud_db > transactions_sampled_10

Documents 59.5K Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#)

ADD DATA EXPORT DATA UPDATE DELETE 25 1 - 25 of 59464

Document 1:

```
{
  "_id": ObjectId('6854e2e9e35a50901889bd59'),
  "age": 2,
  "gender": 0,
  "category": 0,
  "amount_bin": 1,
  "fraud": 0
}
```

Document 2:

```
{
  "_id": ObjectId('6854e2f2e35a5090188a4c17'),
  "age": 2,
  "gender": 0,
  "category": 3,
  "amount_bin": 1,
  "fraud": 0
}
```

Document 3:

```
{
  "_id": ObjectId('6854e2cae35a5090188870da'),
  "age": 1,
  "gender": 0,
  "category": 0,
  "amount_bin": 1,
  "fraud": 0
}
```

Document 4:

```
{
  "_id": ObjectId('6854e280e35a509018857540'),
  "age": 2,
  "gender": 1
}
```

Interfaz del clasificador

Pestaña de Inferencia

InferenciaHipótesis

Instrucciones:

En esta pestaña podrás inferir si se cometió fraude o no para determinada transacción de tarjeta de crédito.

Para esto deberás seleccionar entre distintos valores posibles sobre la transacción.

También deberás seleccionar la hipótesis usada en la inferencia.

Haciendo click en "Clasificar" podrás inferir si se cometió o no fraude.

Hypothesis: Naive Bayes

Gender: F

Age: 2

Category: es_travel

Amount: medium

✓ Clasificar

✓

InferenciaHipótesis

Instrucciones:

En esta pestaña podrás inferir si se cometió fraude o no para determinada transacción de tarjeta de crédito.

Para esto deberás seleccionar entre distintos valores posibles sobre la transacción.

También deberás seleccionar la hipótesis usada en la inferencia.

Haciendo click en "Clasificar" podrás inferir si se cometió o no fraude.

Hypothesis: Naive Bayes

Gender: F

Age: 2

Category: es_travel

Amount: very low

✓ Clasificar

✗ Invalid

(a) Visualización de hipótesis 1

(b) Visualización de hipótesis 2

Figura 1: Comparación de estructuras de hipótesis

Pestaña de Hipótesis

El grafo causal se construye dinámicamente usando **networkx**, permitiendo al usuario interpretar las relaciones causa-efecto establecidas.

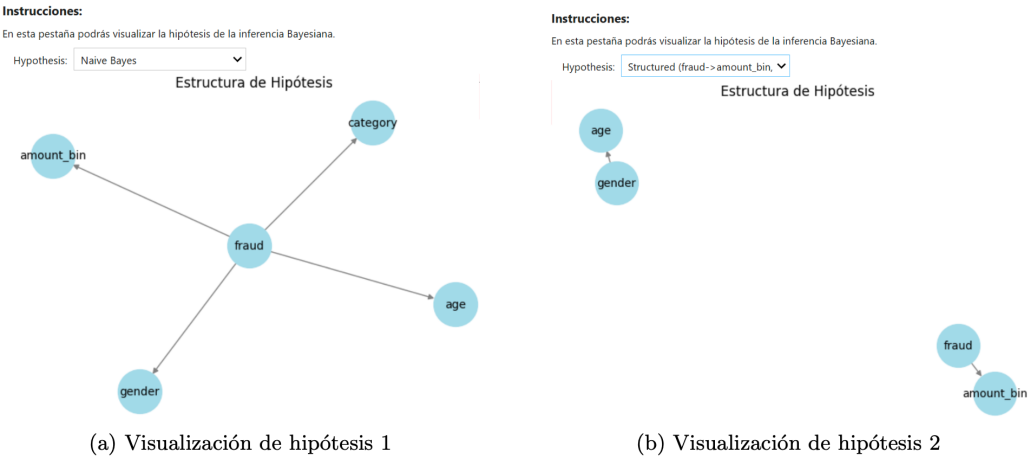


Figura 2: Comparación de estructuras de hipótesis

Benchmarks

Benchmarks de Optimización

Optimizaciones

- 1. Ninguna:** Realiza consultas directas a la base de datos para cada conteo necesario, sin optimizaciones específicas a nivel de la consulta o caché.
 - A pesar de que la base de datos ha sido preprocesada (ej. binning), cada **count_documents** aún puede ser costoso.
 - Cada vez que el clasificador necesita obtener un conteo de documentos para una combinación de valores (p. ej. `count_documents({'gender': 'M', 'age': '2'})`), realiza una consulta completa a la colección en la base de datos.
- 2. Indexes:** Estructuras de datos en la base de datos que permiten búsquedas y recuperación de documentos mucho más rápidas en campos específicos.
 - Mejoran el rendimiento al permitir que la base de datos salte directamente a los datos relevantes en lugar de escanear toda la colección.

```
self.data_collection.create_index([(parent, 1), (var, 1)])
```

- 3. Precomputed database:** Almacenar conteos o agregaciones frecuentes en una colección separada (**precomputed**) dentro de la base de datos.
 - Mejora el rendimiento al evitar cálculos repetitivos en vivo; en su lugar, el sistema realiza una búsqueda rápida en esta tabla de resultados ya calculados.

```
res = self.precomputed.find_one(evidence, {"count": 1})
if res is not None:
    count = res["count"]
else:
    count = self.data_collection.count_documents(evidence)
return count
```

- 4. LRU cache:** Caché en **memoria RAM** que guarda los resultados de las llamadas a funciones.
 - La primera vez que se ejecuta una consulta de conteo, el resultado se almacena; las veces subsiguientes con los mismos argumentos, el resultado se devuelve instantáneamente desde la RAM.

```
@lru_cache(maxsize=10000) # You can adjust maxsize based on expected unique
queries
def _cached_compute_counts(self, evidence_tuple):
    evidence = dict(evidence_tuple) # Convert tuple back to dict
    ...
    return count

def compute_counts(self, evidence):
    # Convert the dictionary (which is not hashable) to a sorted tuple of (key,
value) pairs
    # so it can be used as a cache key.
    hashable_evidence = tuple(sorted(evidence.items()))
    return self._cached_compute_counts(hashable_evidence)
```

Benchmarks de Optimización

Análisis

• **benchmark0:** No optimizaciones

Out[*]=

	test_id	repeat	category	gender	age	amount_bin	fraud_pred	
1	0	0	es_fashion	M	0	medium		
2	0	1	es_fashion	M	0	medium		
3	0	2	es_fashion	M	0	medium		
4	0	3	es_fashion	M	0	medium		
5	0	4	es_fashion	M	0	medium		
6	0	5	es_fashion	M	0	medium		

• **benchmark1:** Solo MongoDB optimización por indexación

Out[*]=

	test_id	repeat	category	gender	age	amount_bin	fraud_pred	
1	0	0	es_fashion	M	0	medium		
2	0	1	es_fashion	M	0	medium		
3	0	2	es_fashion	M	0	medium		
4	0	3	es_fashion	M	0	medium		
5	0	4	es_fashion	M	0	medium		
6	0	5	es_fashion	M	0	medium		

• **benchmark2:** MongoDB index optimization + precomputed counts

Out[*]=

	test_id	repeat	category	gender	age	amount_bin	fraud_pred
5	0	4	es_fashion	M	0	medium	
6	0	5	es_fashion	M	0	medium	
7	0	6	es_fashion	M	0	medium	
8	0	7	es_fashion	M	0	medium	
9	0	8	es_fashion	M	0	medium	
10	0	9	es_fashion	M	0	medium	

• **benchmark3:** mongodb index optimization + precomputed counts + LRU cache

Out[*]=

	test_id	repeat	category	gender	age	amount_bin	fraud_pred
1	0	0	es_fashion	M	0	medium	
2	0	1	es_fashion	M	0	medium	
3	0	2	es_fashion	M	0	medium	
4	0	3	es_fashion	M	0	medium	
5	0	4	es_fashion	M	0	medium	
6	0	5	es_fashion	M	0	medium	

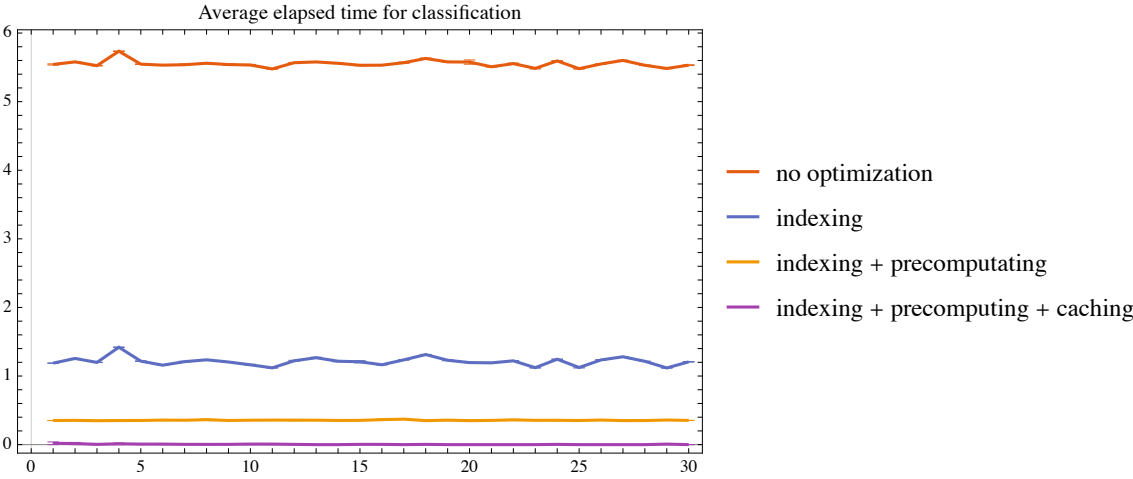
Benchmarks de Optimización

Statistics Summary

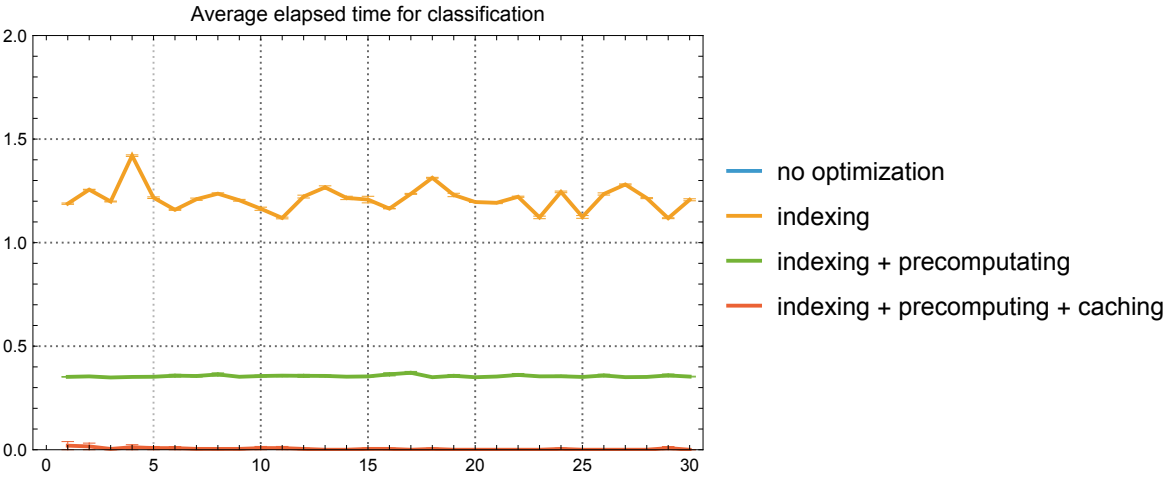
Out[*]=

name	Count	NonNumeric	Mean	StandardDeviation	Min
benchmark 0	300	0	5.55147 s	0.0551509 s	5.45389 s
benchmark 1	300	0	1.21282 s	0.0624999 s	1.10361 s
benchmark 2	300	0	0.355424 s	0.0140068 s	0.342133 s
benchmark 3	300	0	0.00411283 s	0.0199202 s	0.0000157356

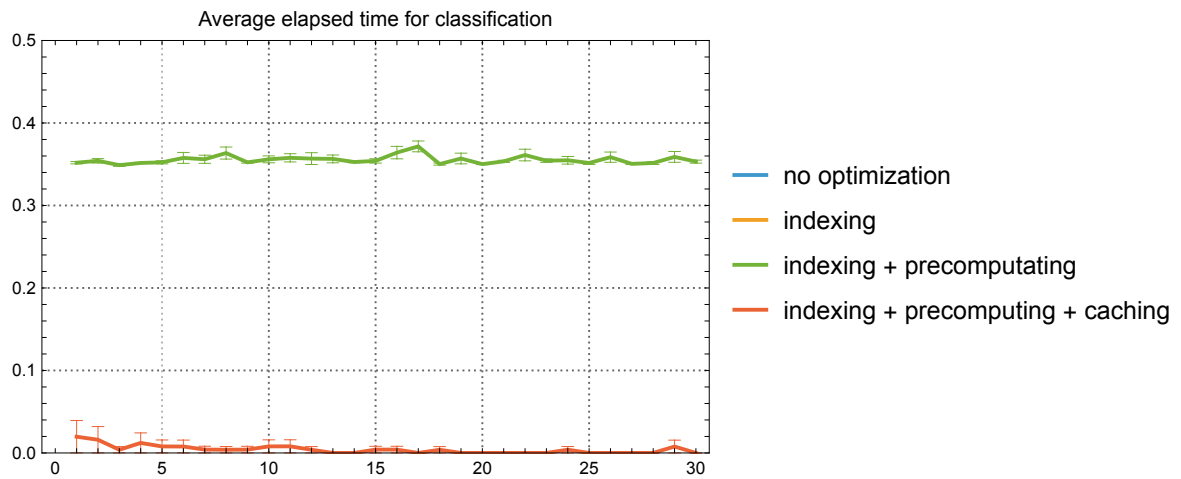
Out[*]=



Out[*]=



Out[^{*}]=



Caching es realmente rápido:

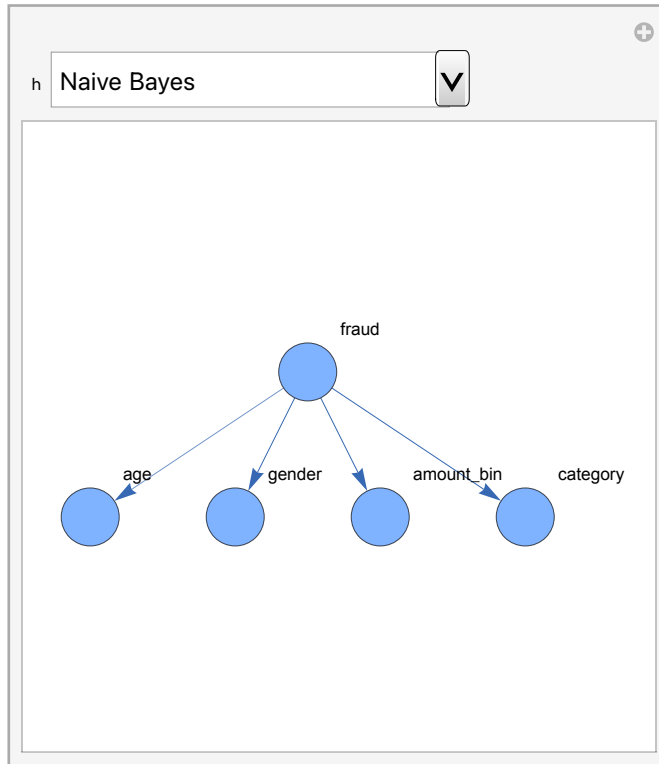
```
In[*]:= UnitConvert[MeanAround[benchmark3[All, "elapsed"], "Milliseconds"]
Out[*]=
(4.1 ± 1.2) ms
```

Benchmarks de Hipótesis

Hipótesis

Hipótesis determinadas a mano (leer comentarios en la sección de K2):

Out[⁸]=



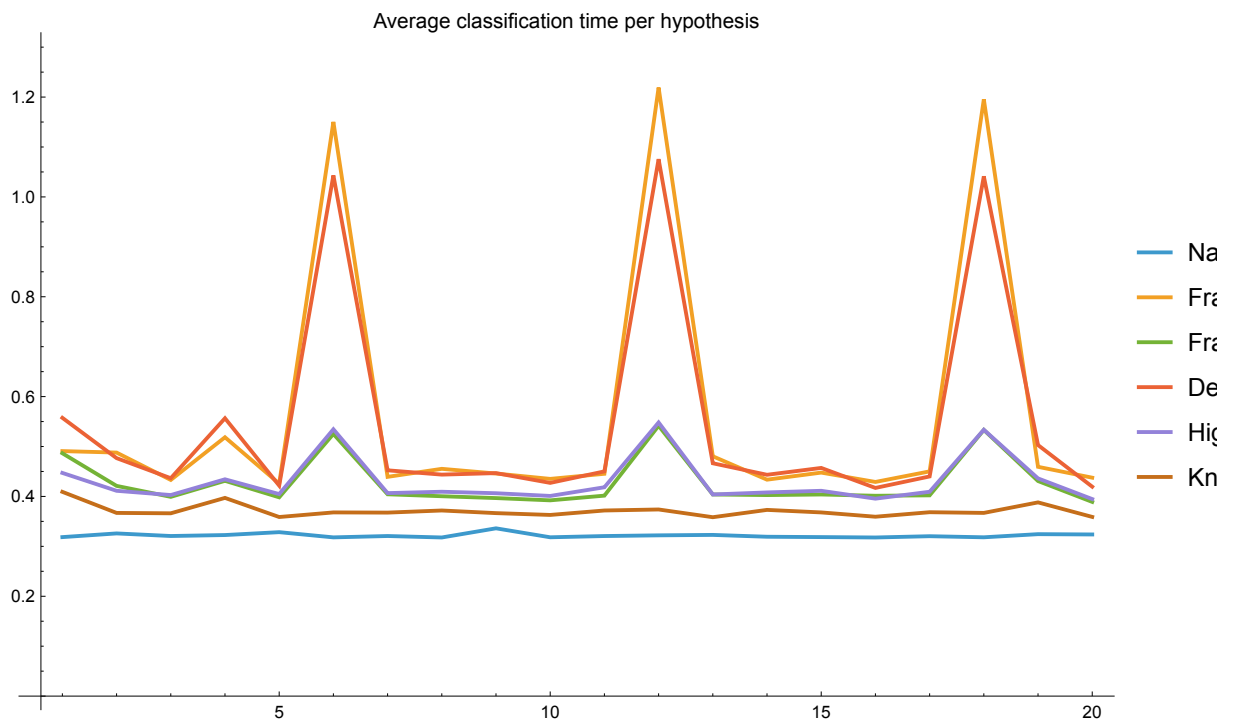
Análisis

Out[*]=

h Naive Bayes

	evidence_id	avg_time_sec (s)	fraud_prediction	probability	age	gender
1	1	0.319	False	0.0018	3	M
2	2	0.326	False	0.0008	4	F
3	3	0.321	False	0.0000	4	M
4	4	0.323	False	0.0029	4	F
5	5	0.328	True	0.0000	5	F
6	6	0.318	False	0.0066	5	F

Out[*]=



Benchmarks de Tamaño de datos

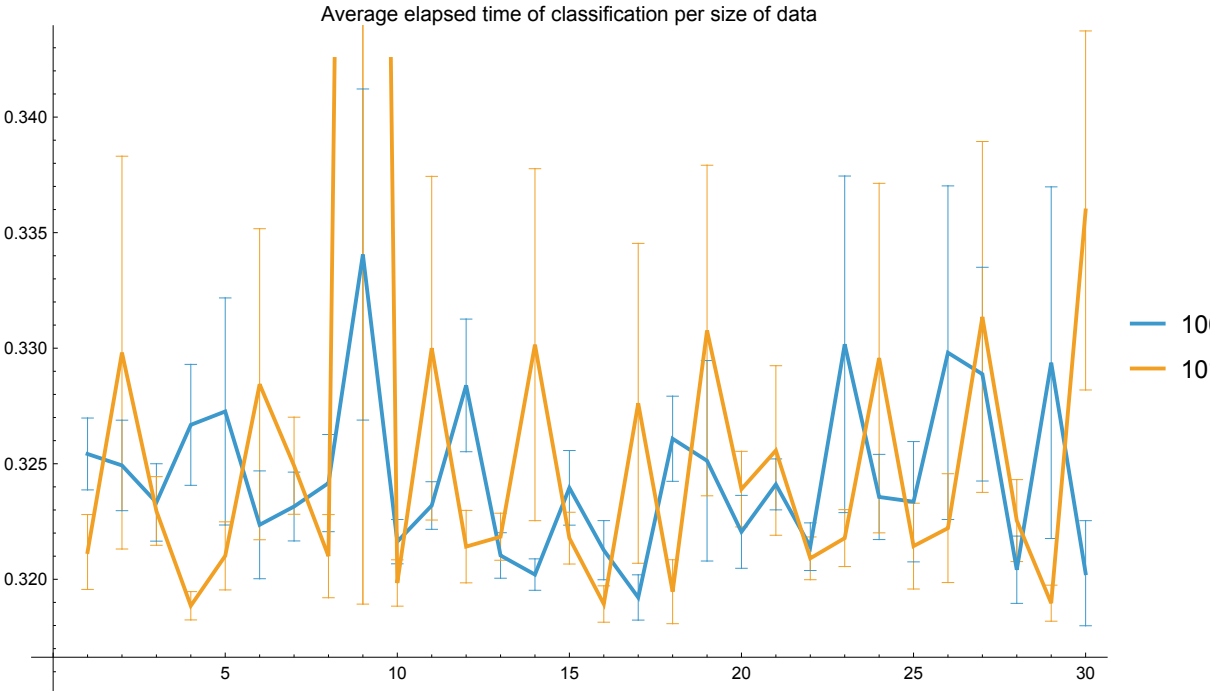
Out[*]=

f

transactions_indexedtransactions_sampled_10

	test_id	repeat	category	gender	age	amount_bin	
1	0	0	es_fashion	M	0	medium	
2	0	1	es_fashion	M	0	medium	
3	0	2	es_fashion	M	0	medium	
4	0	3	es_fashion	M	0	medium	
5	0	4	es_fashion	M	0	medium	
6	0	5	es_fashion	M	0	medium	

Out[*]=



Métricas de Clasificación

Matriz de confusión:

- **Verdadero Positivo (VP)**: El modelo predijo que era fraude, y realmente era fraude.
- **Verdadero Negativo (VN)**: El modelo predijo que NO era fraude, y realmente NO era fraude.
- **Falso Positivo (FP)**: El modelo predijo que era fraude, pero en realidad NO era fraude.
- **Falso Negativo (FN)**: El modelo predijo que NO era fraude, pero en realidad SÍ era fraude.

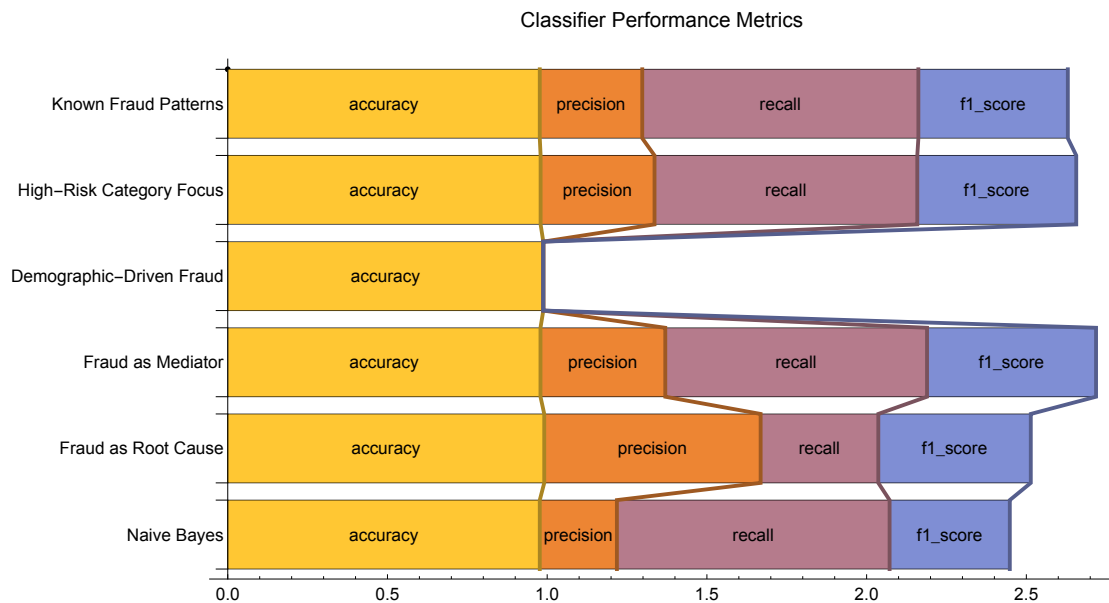
Out[*] =

model Naive Bayes			
		Actually Positive	Actually Negative
Predicted Positive		35 (TP)	110 (FP)
Predicated Negative		6 (FN)	4849 (TN)

Otras Métricas

- **Accuracy:** $(VP + VN) / (VP + VN + FP + FN)$
 - Es la proporción de predicciones correctas (tanto fraudes como no-fraudes) sobre el total de predicciones.
- **Precision:** $VP / (VP + FP)$
 - De todas las veces que el modelo predijo “fraude”, ¿cuántas veces realmente fue fraude?
- **Recall (Exhaustividad o Sensibilidad):** $VP / (VP + FN)$
 - De todos los casos que realmente eran fraude, ¿cuántos logró detectar el modelo?
- **F1-Score:** $2 \times (Precision \times Recall) / (Precision + Recall)$
 - Es un promedio “armónico” de la Precisión y el Recall. Proporciona un equilibrio entre ambas métricas.

Out[*]=



En resumen, para la detección de fraude:

- **Recall** es a menudo lo más importante porque no quieres que los fraudes se te escapen.
- **Precision** también es importante para evitar la “fatiga de alertas” o molestar a clientes legítimos.
- **F1-Score** busca un buen compromiso entre ambos.
- Accuracy puede ser engañosa y debe usarse con precaución, especialmente si el número de fraudes es muy pequeño.

Algoritmo K2 (learn_k2_structures.py)

Objetivo: descubrir automáticamente cómo se relacionan las diferentes variables en el dataset de transacciones para construir un modelo de red bayesiana.

- Para lograrlo, inicializa un clasificador bayesiano.
- Luego se utiliza el algoritmo **K2** para analizar los datos y encontrar las conexiones más probables entre las variables
 - E.g., determinar si la edad o el género influyen en la categoría de una transacción o en la probabilidad de fraude.
- Este proceso se repite varias veces, probando con diferentes límites para la cantidad de “padres” que cada variable puede tener.

Out[*] =

metadata	dataset_size	594 643
	variables	{age, gender,
	cardinalities	< age → 8, ge
	learning_timestamp	1 751 895 502.
hypotheses	Naive Bayes	< fraud → {ag
	Structured (fraud→amount_bin, gender→age)	< fraud → {an
	K2 learned (u=1)	< >
	K2 learned (u=2)	< >
	K2 learned (u=3)	< >
	7 total ›	
learning_details	K2 learned (u=1)	< structure →
	K2 learned (u=2)	< structure →
	K2 learned (u=3)	< structure →
	K2 learned (u=4)	< structure →
	K2 learned (u=5)	< structure →

Algoritmo K2 (learn_k2_structures.py)

No es raro que el algoritmo K2 encuentre que algunas (o todas) las variables no tienen padres.

No necesariamente significa que haya un error:

- **No hay dependencias fuertes:** Si en los datos una variable realmente no depende mucho de otras, K2 lo detecta y no le asigna padres
- **Datos limitados:** Con pocos datos, es difícil ver relaciones claras, y el algoritmo prefiere simplicidad
 - Este no es el caso con *fraud_credit_card.xlsx*
- **Orden de las variables:** El orden en que le das las variables a K2 es crucial.
 - Si un posible padre aparece después de su hijo, K2 nunca lo considerará
- **El parámetro alpha:** Este valor afecta qué tan “convencido” está el algoritmo de que hay una relación
 - Un alpha alto puede hacer que el modelo prefiera no asignar padres
 - Se probaron $\alpha=0.01$, $\alpha=0.1$, $\alpha=1.0$, $\alpha=10.0$, $\alpha=100.0$ sin resultados
- **Datos dispersos:** Si las variables tienen muchos valores posibles y pocos ejemplos para cada combinación, K2 puede tener dificultades para encontrar patrones