**Nombre:** Jéssica Ñauta
**Carrera:** Ingeniería de Sistemas.

# KNN CLASIFICACIÓN DE MIEMBROS DEL CONGRESO UTILIZANDO ALGORITMOS DE SIMILITUD EN NEO4J

## CONJUNTO DE DATOS

Este conjunto de datos incluye votos para cada uno de los congresistas de la Cámara de Representantes de los Estados Unidos sobre los 16 votos clave identificados por la CQA. La CQA enumera nueve tipos diferentes de votos: votó, emparejó y anunció (estos tres se simplificaron a sí), votó en contra, emparejó en contra y anunció en contra (estos tres se simplificaron en contra), votó presente, votó presente para evitar conflicto de intereses, y no votó ni dio a conocer una posición (estos tres se simplificaron a una disposición desconocida).

## Código:

LOAD CSV FROM "http://archive.ics.uci.edu/ml/machine-learning-databases/voting-records/house-votes-84.data" as row

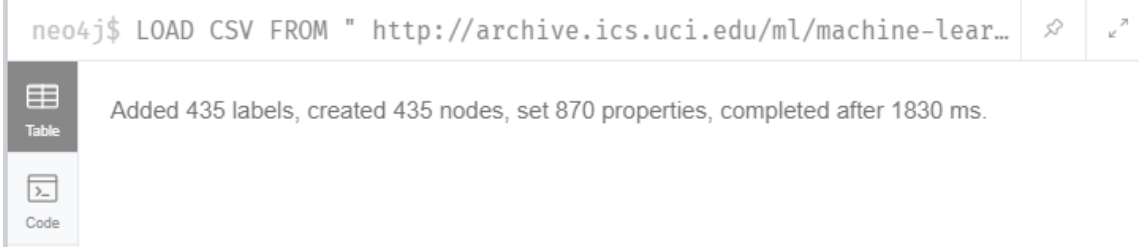
CREATE (p:Person)

SET p.class = row[0],

p.features = row[1..];

```
1 LOAD CSV FROM " http://archive.ics.uci.edu/ml/machine-
  learning-databases/voting-records/house-votes-84.data " as
  row
2 CREATE (p: Persona)
3 SET p.class = row [0],
4     p.features = row [1 .. ];
```

## Resultado:



## VOTOS FALTANTES

Veamos cuántos miembros del congreso tienen al menos un voto faltante.

## Código:

MATCH (n:Person)

WHERE "?" in n.features

RETURN count(n)

```
MATCH (n:Person)
WHERE "?" in n.features
RETURN count(n)
```

## Resultado:

```
neo4j$ MATCH (n:Person) WHERE "?" in n.features RETURN count…
```

Table

| count(n) |
| --- |
| 203 |

Text

Code

Casi la mitad de los miembros del congreso tienen votos faltantes. Eso es bastante significativo, así que profundicemos más. Revisaremos cuál es la distribución de los votos faltantes por miembro.

## Código:

MATCH (p:Person)

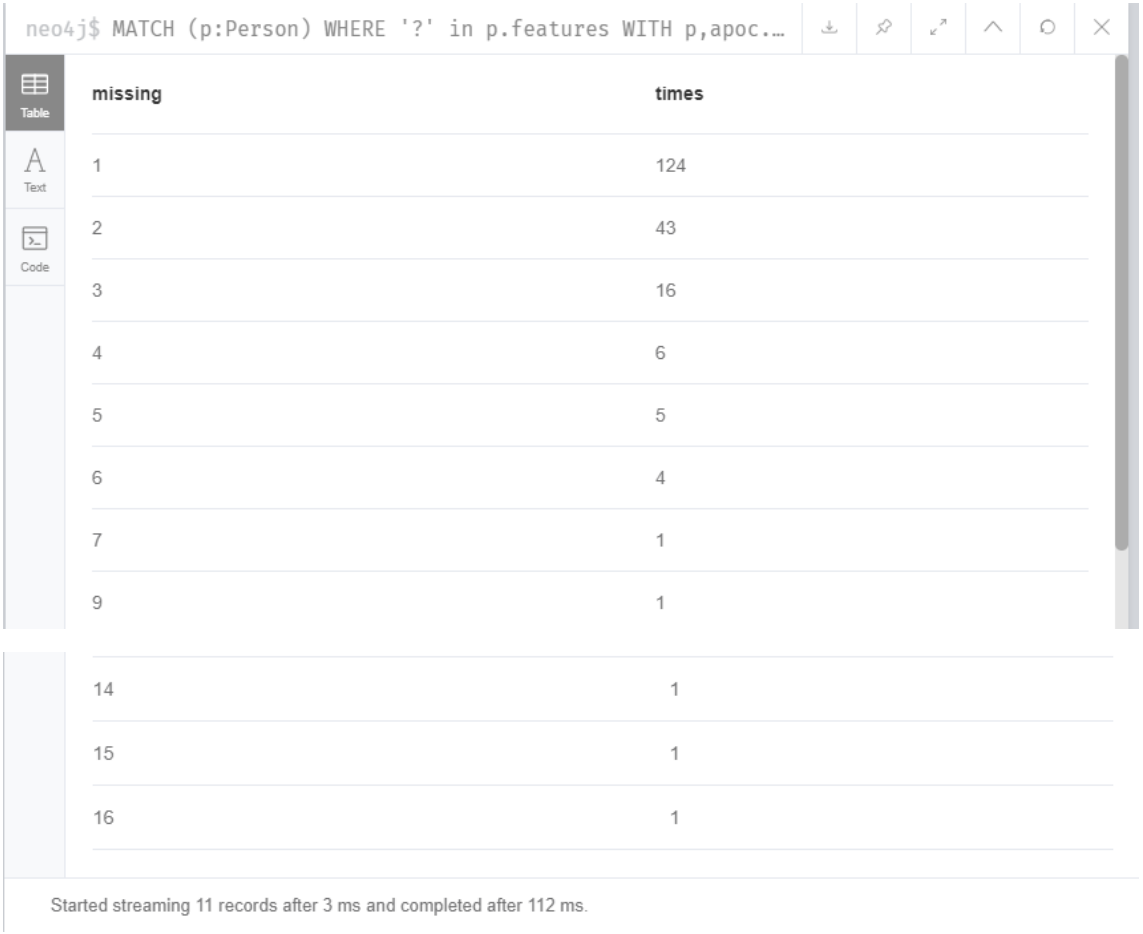
WHERE '?' in p.features

WITH p,apoc.coll.occurrences(p.features,'?') as missing

RETURN missing,count(*) as times ORDER BY missing ASC

```
MATCH (p:Person)
WHERE '?' in p.features
WITH p,apoc.coll.occurrences(p.features,'?') as missing
RETURN missing,count(*) as times ORDER BY missing ASC
```

## Resultado:




Tres miembros casi nunca votaron (14,15,16 votos faltantes) y dos de ellos (7,8 votos faltantes) tienen más del 50% de votos faltantes. Los excluiremos de nuestro análisis posterior para intentar reducir el ruido.

## Código:

MATCH (p:Person)

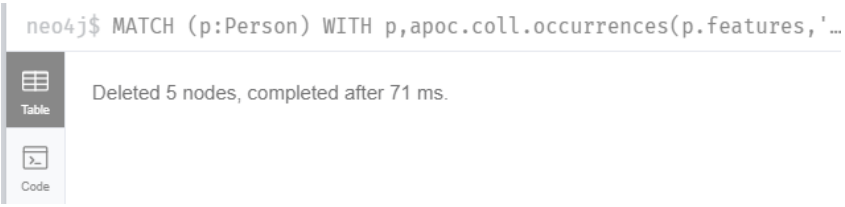
WITH p,apoc.coll.occurrences(p.features,'?') as missing

WHERE missing > 6

DELETE p

```
MATCH (p:Person)
WITH p,apoc.coll.occurrences(p.features,'?') as missing
WHERE missing > 6
DELETE p
```

## Resultado:




## DATOS DE ENTRENAMIENTO Y PRUEBA

Dividamos nuestro conjunto de datos en dos subconjuntos, donde el 80% de los nodos se marcarán como datos de entrenamiento y el 20% restante como datos de prueba. Hay un total de 430 nodos en nuestro gráfico. Marcaremos 344 nodos como subconjunto de entrenamiento y el resto como prueba.

## Marcar datos de entrenamiento

## Código:

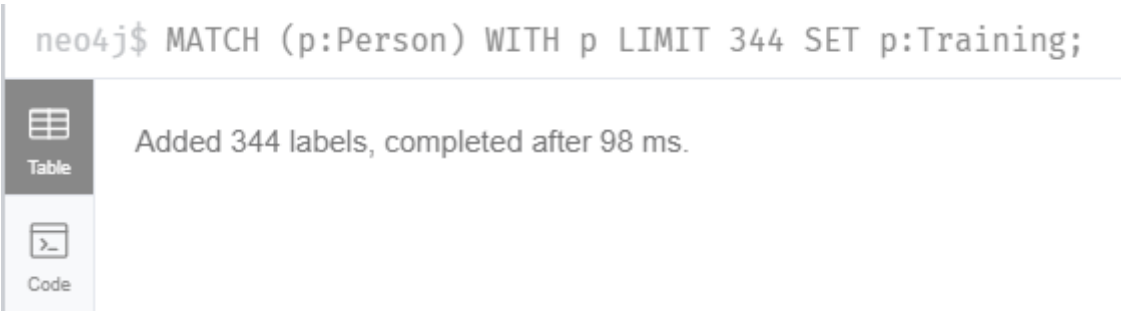
MATCH (p:Person)

WITH p LIMIT 344

SET p:Training;

```
1 MATCH (p:Person)
2 WITH p LIMIT 344
3 SET p:Training;
```

## Resultado:



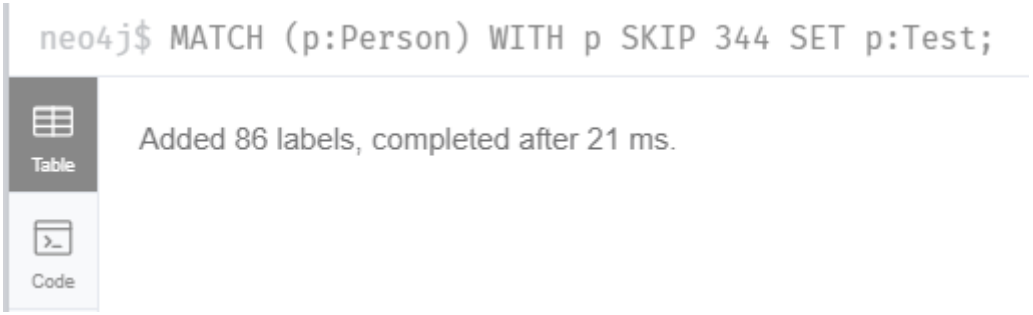

## Marcar datos de prueba

## Código:

MATCH (p:Person)
WITH p SKIP 344
SET p:Test;

```
MATCH (p:Person)
WITH p SKIP 344
SET p:Test;
```

**Resultado:**




## Vector de características

**Hay tres valores posibles en los conjuntos de características. Los mapearemos de la siguiente manera:**

**"Y" a 1**

**"N" a 0**

**"?" a 0.5**

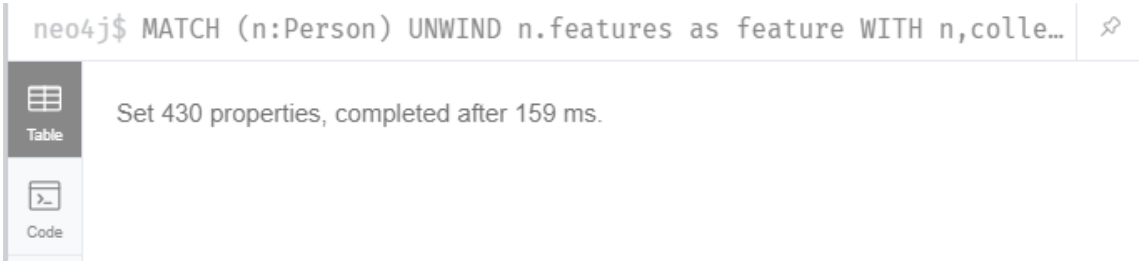
## Transformar a vector de características

## Código:

```
MATCH (n:Person)
UNWIND n.features as feature
WITH n,collect(CASE feature WHEN 'y' THEN 1
                            WHEN 'n' THEN 0
                             ELSE 0.5 END) as feature_vector
SET n.feature_vector = feature_vector
```

```
MATCH (n:Person)
UNWIND n.features as feature
WITH n,collect(CASE feature WHEN 'y' THEN 1
                            WHEN 'n' THEN 0
                            ELSE 0.5 END) as feature_vector
SET n.feature_vector = feature_vector
```

**Resultado:**




**Ejemplo de transformación de conjunto de características a vector de características**

| features | feature_vector |
|---|---|
| [“n”, “y”, “n”, “y”, “y”, “n”, “n”, “n”, “n”, “n”, “?”, “?”, “y”, “y”, “n”, “n”] | [0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0.5, 0.5, 1, 1, 0, 0] |

# ALGORITMO CLASIFICADOR KNN

Usaremos la distancia euclidiana como la función de distancia y el valor topK de 3. Es aconsejable usar un número impar como K para evitar producir casos extremos, donde, por ejemplo, con los dos vecinos superiores y cada uno con una clase diferente, terminamos sin clase mayoritaria, pero una división 50/50 entre los dos.

Tenemos una situación específica en la que queremos comenzar con todos los nodos etiquetados como Prueba y encontrar los tres nodos vecinos principales solo del subconjunto de Entrenamiento. De lo contrario, todos los nodos etiquetados para Prueba también se considerarían parte de los datos de Entrenamiento, que es algo que queremos evitar. Esta es exactamente la razón por la cual no podemos usar una consulta simple como:

CALL algo.similarity.euclidean(data, {topK: 3, write:true})

Tendremos que usar apoc.cypher.run para limitar los resultados de las coincidencias por fila en lugar de por consulta. Esto nos ayudará a recuperar los 3 primeros vecinos por nodo. Encuentre más en la base de conocimiento .

También usaremos una combinación de apoc.coll.frequenciesy apoc.coll.sortMapspara recuperar el elemento más frecuente en una colección.

apoc.coll.sortMaps (apoc.coll.frequencies (colección ), '^ count') [- 1]

## Consulta

### Código:

MATCH (test:Test)

WITH test,test.feature_vector as feature_vector

CALL apoc.cypher.run('MATCH (training:Training)

WITH training,gds.alpha.similarity.euclideanDistance($feature_vector,

training.feature_vector) AS similarity

ORDER BY similarity ASC LIMIT 3

RETURN collect(training.class) as classes',

{feature_vector:feature_vector}) YIELD value

WITH test.class as class, apoc.coll.sortMaps(apoc.coll.frequencies(value.classes), '^count')[-

1].item as predicted_class

WITH sum(CASE when class = predicted_class THEN 1 ELSE 0 END) as correct_predictions,

count(*) as total_predictions

RETURN correct_predictions,total_predictions, correct_predictions / toFloat(total_predictions)

as ratio;

```
MATCH (test:Test)
WITH test,test.feature_vector as feature_vector
CALL apoc.cypher.run('MATCH(training:Training)
    // calculate euclidian distance between each test node
and all training nodes
    WITH
training,algo.similarity.euclideanDistance($feature_vector,
training.feature_vector) AS similarity
    // return only top 3 nodes
    ORDER BY similarity ASC LIMIT 3
    RETURN collect(training.class) as classes',
    {feature_vector:feature_vector}) YIELD value
WITH test.class as class,
apoc.coll.sortMaps(apoc.coll.frequencies(value.classes),
'^count')[-1].item as predicted_class
WITH sum(CASE when class = predicted_class THEN 1 ELSE 0
END) as correct_predictions, count(*) as total_predictions
RETURN correct_predictions,total_predictions,
correct_predictions / toFloat(total_predictions) as ratio
```

## Resultado:

```
neo4j$ MATCH (test:Test) WITH test,test.feature_vector as feature_vector CALL a…
```

Table | Text | Code

| correct_predictions | total_predictions | ratio |
|---|---|---|
| 77 | 86 | 0.8953488372093024 |

Started streaming 1 records after 1 ms and completed after 577 ms.