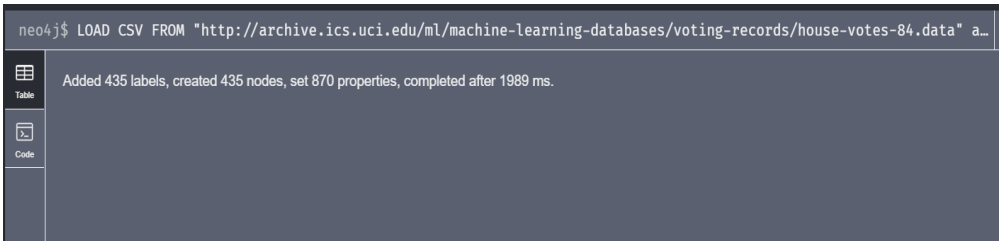
Nombre: Erika Morocho

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).

1. Cargamos el CVS.

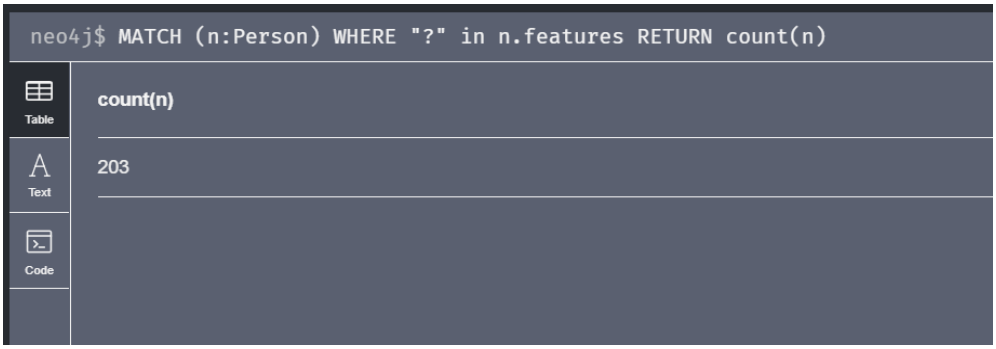
```
LOAD CSV FROM "http://archive.ics.uci.edu/ml/machine-learning-databases/voting-records/house-votes-84.data" as row
CREATE (p:Person)
SET p.class = row[0],
  p.features = row[1..]
```



2. Votos Faltantes

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

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



2.2 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.

```
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
```

```
o4j$ MATCH (p:Person) WHERE '?' in p.features WITH p,apoc.coll.occurrences(p.features,'?') as missing RETURN missing,
```

missing	times
1	124
2	43
3	16
4	6
5	5
6	4
7	1
9	1
14	1
15	1
16	1

2.3 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.

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

```
neo4j$ MATCH (p:Person) WITH p,apoc.coll.occurrences(p.features,'?') as missing WHERE missing > 6 DELETE p
Table
Code
Deleted 5 nodes, completed after 18 ms.
```

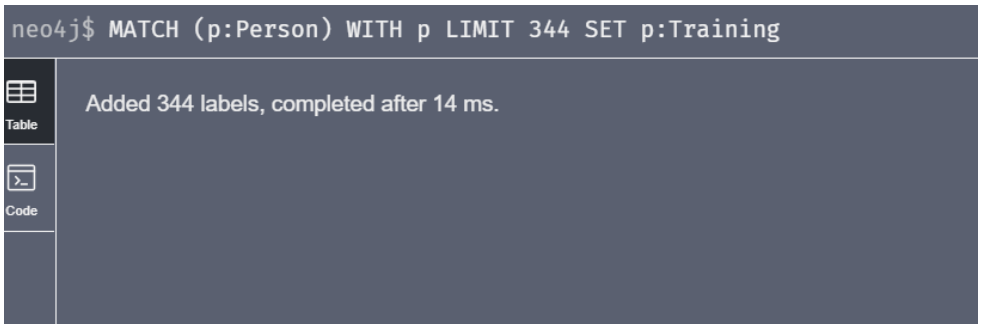
3. Datos de entrenamiento y prueba

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.

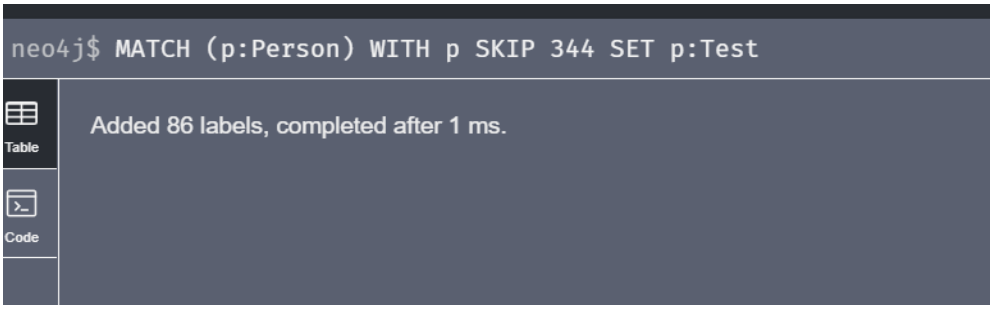
3.1 Marcar datos de entrenamiento

```
MATCH (p:Person)
WITH p LIMIT 344
SET p:Training
```



3.2 Marcar datos de prueba

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



4. Transformar a vector de características

```
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
```

5. 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.

```
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
```