ADT\_ WEEK\_11

1. How many reviews does each Matrix movie have?

MATCH (m:Movie)<-[:RATED]-(u:User)

WHERE m.title CONTAINS 'Matrix'

WITH m, count(\*) AS reviews

RETURN m.title AS movie, reviews

ORDER BY reviews DESC LIMIT 5;

1. Content-Based Filtering

MATCH p=(m:Movie {title: 'Net, The'})

-[:ACTED\_IN|IN\_GENRE|DIRECTED\*2]-()

RETURN p LIMIT 25

1. Collaborative Filtering

MATCH (m:Movie {title: 'Crimson Tide'})<-[:RATED]-

(u:User)-[:RATED]->(rec:Movie)

WITH rec, COUNT(\*) AS usersWhoAlsoWatched

ORDER BY usersWhoAlsoWatched DESC LIMIT 25

RETURN rec.title AS recommendation, usersWhoAlsoWatched

1. Content-Based Filtering

MATCH (m:Movie)-[:IN\_GENRE]->(g:Genre)

<-[:IN\_GENRE]-(rec:Movie)

WHERE m.title = 'Inception'

WITH rec, collect(g.name) AS genres, count(\*) AS commonGenres

RETURN rec.title, genres, commonGenres

ORDER BY commonGenres DESC LIMIT 10;

1. Personalized Recommendations Based on Genres

MATCH (u:User {name: 'Angelica Rodriguez'})-[r:RATED]->(m:Movie),

(m)-[:IN\_GENRE]->(g:Genre)<-[:IN\_GENRE]-(rec:Movie)

WHERE NOT EXISTS{ (u)-[:RATED]->(rec) }

WITH rec, g.name as genre, count(\*) AS count

WITH rec, collect([genre, count]) AS scoreComponents

RETURN rec.title AS recommendation, rec.year AS year, scoreComponents,

reduce(s=0,x in scoreComponents | s+x[1]) AS score

ORDER BY score DESC LIMIT 10

1. Weighted Content Algorithm

MATCH (m:Movie) WHERE m.title = 'Wizard of Oz, The'

MATCH (m)-[:IN\_GENRE]->(g:Genre)<-[:IN\_GENRE]-(rec:Movie)

WITH m, rec, count(\*) AS gs

OPTIONAL MATCH (m)<-[:ACTED\_IN]-(a)-[:ACTED\_IN]->(rec)

WITH m, rec, gs, count(a) AS as

OPTIONAL MATCH (m)<-[:DIRECTED]-(d)-[:DIRECTED]->(rec)

WITH m, rec, gs, as, count(d) AS ds

RETURN rec.title AS recommendation,

(5\*gs)+(3\*as)+(4\*ds) AS score

ORDER BY score DESC LIMIT 25

1. Content-Based Similarity Metrics

# query 1: MATCH (m:Movie {title:'Inception'})-[:IN\_GENRE]->

(g:Genre)<-[:IN\_GENRE]-(other:Movie)

WITH m, other, count(g) AS intersection, collect(g.name) as common

WITH m,other, intersection, common,

[(m)-[:IN\_GENRE]->(mg) | mg.name] AS set1,

[(other)-[:IN\_GENRE]->(og) | og.name] AS set2

WITH m,other,intersection, common, set1, set2,

set1+[x IN set2 WHERE NOT x IN set1] AS union

RETURN m.title, other.title, common, set1,set2,

((1.0\*intersection)/size(union)) AS jaccard

ORDER BY jaccard DESC LIMIT 25

# query2:

MATCH (m:Movie {title: 'Inception'})-[:IN\_GENRE|ACTED\_IN|DIRECTED]-

(t)<-[:IN\_GENRE|ACTED\_IN|DIRECTED]-(other:Movie)

WITH m, other, count(t) AS intersection, collect(t.name) AS common,

[(m)-[:IN\_GENRE|ACTED\_IN|DIRECTED]-(mt) | mt.name] AS set1,

[(other)-[:IN\_GENRE|ACTED\_IN|DIRECTED]-(ot) | ot.name] AS set2

WITH m,other,intersection, common, set1, set2,

set1 + [x IN set2 WHERE NOT x IN set1] AS union

RETURN m.title, other.title, common, set1,set2,

((1.0\*intersection)/size(union)) AS jaccard

ORDER BY jaccard DESC LIMIT 25

1. Collaborative Filtering – Leveraging Movie Ratings

MATCH (u:User {name: 'Misty Williams'})

MATCH (u)-[r:RATED]->(m:Movie)

RETURN \*

LIMIT 100;

#average rating

MATCH (u:User {name: 'Misty Williams'})

MATCH (u)-[r:RATED]->(m:Movie)

RETURN avg(r.rating) AS average;

1. Movies rating more than average

// What are the movies that Misty liked more than average?

MATCH (u:User {name: 'Misty Williams'})

MATCH (u)-[r:RATED]->(m:Movie)

WITH u, avg(r.rating) AS average

MATCH (u)-[r:RATED]->(m:Movie)

WHERE r.rating > average

RETURN \*

LIMIT 100;

1. Collaborative Filtering – The Wisdom of Crowds

MATCH (u:User {name: 'Cynthia Freeman'})-[:RATED]->

(:Movie)<-[:RATED]-(peer:User)

MATCH (peer)-[:RATED]->(rec:Movie)

WHERE NOT EXISTS { (u)-[:RATED]->(rec) }

RETURN rec.title, rec.year, rec.plot

LIMIT 25

MATCH (u:User {name: 'Cynthia Freeman'})-[r1:RATED]->

(:Movie)<-[r2:RATED]-(peer:User)

WHERE abs(r1.rating-r2.rating) < 2 // similarly rated

WITH distinct u, peer

MATCH (peer)-[r3:RATED]->(rec:Movie)

WHERE r3.rating > 3

AND NOT EXISTS { (u)-[:RATED]->(rec) }

WITH rec, count(\*) as freq, avg(r3.rating) as rating

RETURN rec.title, rec.year, rating, freq, rec.plot

ORDER BY rating DESC, freq DESC

LIMIT 25

# Only Consider Genres Liked by the User

// compute mean rating

MATCH (u:User {name: 'Andrew Freeman'})-[r:RATED]->(m:Movie)

WITH u, avg(r.rating) AS mean

// find genres with higher than average rating and their number of rated movies

MATCH (u)-[r:RATED]->(m:Movie)

-[:IN\_GENRE]->(g:Genre)

WHERE r.rating > mean

WITH u, g, count(\*) AS score

// find movies in those genres, that have not been watched yet

MATCH (g)<-[:IN\_GENRE]-(rec:Movie)

WHERE NOT EXISTS { (u)-[:RATED]->(rec) }

// order by sum of scores

RETURN rec.title AS recommendation, rec.year AS year,

sum(score) AS sscore,

collect(DISTINCT g.name) AS genres

ORDER BY sscore DESC LIMIT 10

1. Collaborative Filtering – Similarity Metrics

// Most similar users using Cosine similarity

MATCH (p1:User {name: "Cynthia Freeman"})-[x:RATED]->

(m:Movie)<-[y:RATED]-(p2:User)

WITH p1, p2, count(m) AS numbermovies,

sum(x.rating \* y.rating) AS xyDotProduct,

collect(x.rating) as xRatings, collect(y.rating) as yRatings

WHERE numbermovies > 10

WITH p1, p2, xyDotProduct,

sqrt(reduce(xDot = 0.0, a IN xRatings | xDot + a^2)) AS xLength,

sqrt(reduce(yDot = 0.0, b IN yRatings | yDot + b^2)) AS yLength

RETURN p1.name, p2.name, xyDotProduct / (xLength \* yLength) AS sim

ORDER BY sim DESC

LIMIT 100;

MATCH (p1:User {name: 'Cynthia Freeman'})-[x:RATED]->(movie)<-[x2:RATED]-(p2:User)

WHERE p2 <> p1

WITH p1, p2, collect(x.rating) AS p1Ratings, collect(x2.rating) AS p2Ratings

WHERE size(p1Ratings) > 10

RETURN p1.name AS from,

p2.name AS to,

gds.similarity.cosine(p1Ratings, p2Ratings) AS similarity

ORDER BY similarity DESC

1. Collaborative Filtering – Similarity Metrics

# Pearson Similarity

MATCH (u1:User {name:"Cynthia Freeman"})-[r:RATED]->(m:Movie)

WITH u1, avg(r.rating) AS u1\_mean

MATCH (u1)-[r1:RATED]->(m:Movie)<-[r2:RATED]-(u2)

WITH u1, u1\_mean, u2, collect({r1: r1, r2: r2}) AS ratings

WHERE size(ratings) > 10

MATCH (u2)-[r:RATED]->(m:Movie)

WITH u1, u1\_mean, u2, avg(r.rating) AS u2\_mean, ratings

UNWIND ratings AS r

WITH sum( (r.r1.rating-u1\_mean) \* (r.r2.rating-u2\_mean) ) AS nom,

sqrt( sum( (r.r1.rating - u1\_mean)^2) \* sum( (r.r2.rating - u2\_mean) ^2)) AS denom,

u1, u2 WHERE denom <> 0

RETURN u1.name, u2.name, nom/denom AS pearson

ORDER BY pearson DESC LIMIT 100

MATCH (p1:User {name: 'Cynthia Freeman'})-[x:RATED]->(movie)<-[x2:RATED]-(p2:User)

WHERE p2 <> p1

WITH p1, p2, collect(x.rating) AS p1Ratings, collect(x2.rating) AS p2Ratings

WHERE size(p1Ratings) > 10

RETURN p1.name AS from,

p2.name AS to,

gds.similarity.pearson(p1Ratings, p2Ratings) AS similarity

ORDER BY similarity DESC

1. KNN based recommendations.

MATCH (u1:User {name:"Cynthia Freeman"})-[r:RATED]->(m:Movie)

WITH u1, avg(r.rating) AS u1\_mean

MATCH (u1)-[r1:RATED]->(m:Movie)<-[r2:RATED]-(u2)

WITH u1, u1\_mean, u2, COLLECT({r1: r1, r2: r2}) AS ratings WHERE size(ratings) > 10

MATCH (u2)-[r:RATED]->(m:Movie)

WITH u1, u1\_mean, u2, avg(r.rating) AS u2\_mean, ratings

UNWIND ratings AS r

WITH sum( (r.r1.rating-u1\_mean) \* (r.r2.rating-u2\_mean) ) AS nom,

sqrt( sum( (r.r1.rating - u1\_mean)^2) \* sum( (r.r2.rating - u2\_mean) ^2)) AS denom,

u1, u2 WHERE denom <> 0

WITH u1, u2, nom/denom AS pearson

ORDER BY pearson DESC LIMIT 10

MATCH (u2)-[r:RATED]->(m:Movie) WHERE NOT EXISTS( (u1)-[:RATED]->(m) )

RETURN m.title, SUM( pearson \* r.rating) AS score

ORDER BY score DESC LIMIT 25