**Problem 1: Topic Modeling**

From the game description data, is it possible to generate some general topics to represent each game? By connecting with other attributes of the game, what is the connection between the topic and other aspects?

To answer this question, we chose to perform text analysis to the description data. We chose LDA model to generate topics. Then, we created a node called Topic in neo4j and connected it to the game node.

Before the modeling, we had to process the data:

For instance, deleting stop words.

A screen shot of a computer program

Description automatically generated

Then, we created functions of embeddings and fitting the LDA model:

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Then, we created a function to import the generated result to neo4j and connect the topic to the game node.

A screen shot of a computer program

Description automatically generated

These are generated topics:

Although it may not have good representation for each description, we still find some useful information from the generated topics.

A screenshot of a computer

Description automatically generated

This is the relationship in neo4j:

A screenshot of a video game

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Homesick is a puzzle game. As we can find in the graphs above, the topic gives some insight of the game.

**Problem 2 New Game Recommendation**

Assume now we have a new game, is it possible to predict the positive ratio of this new game by comparing its attributes to similar games?

To answer the question, I first generated a random new game, then calculated the defined similarity between this new game and other games, then I used the weighted average to calculate the expected positive ratio of this new game. At last, I imported all the used game nodes and the new game node to neo4j to see the relation.

First I generated a random game using the defined function. All the attributes here are selected from the scope of the games.csv dataset.

1. **def** generate\_random\_game():
2. """ Generate a random game entry with similar structure to the original dataset. """
3. **return** {
4. 'app\_id': random.randint(100000, 999999),  # Randomly generated application ID
5. 'title': "Random Game " + str(random.randint(1, 1000)),  # Random title
6. 'date\_release': generate\_date(),
7. 'win': generate\_boolean(),
8. 'mac': generate\_boolean(),
9. 'linux': generate\_boolean(),
10. 'rating': generate\_rating(),
11. 'positive\_ratio': generate\_positive\_ratio(),
12. 'user\_reviews': generate\_user\_reviews(),
13. 'price\_final': generate\_price(),
14. 'price\_original': generate\_price(),
15. 'discount': generate\_discount(),
16. 'steam\_deck': generate\_boolean()
17. }

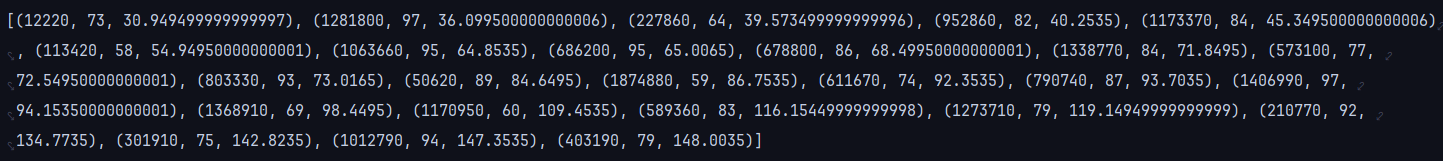
Then I inserted the random game to postgres.

1. insert\_query = """
2. INSERT INTO games (app\_id, title, win, mac, linux, user\_reviews, price\_final, price\_original, discount, steam\_deck)
3. VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s);
4. """
5. **def** insert\_game(query, random\_game, pg\_cursor, pg\_conn):
6. # execute the query
7. pg\_cursor.execute(query, (
8. random\_game['app\_id'],
9. random\_game['title'],
10. random\_game['win'],
11. random\_game['mac'],
12. random\_game['linux'],
13. random\_game['user\_reviews'],
14. random\_game['price\_final'],
15. random\_game['price\_original'],
16. random\_game['discount'],
17. random\_game['steam\_deck']
18. ))
20. pg\_conn.commit()

Then I calculate the difference between the new game and all existing games using the postgres query:

1. similarity\_query = """
2. SELECT g.app\_id, g.positive\_ratio,
3. (ABS(g.win::int - %s::int) \* 5 +
4. ABS(g.mac::int - %s::int) \* 5 +
5. ABS(g.linux::int - %s::int) \* 3 +
6. ABS(g.user\_reviews - %s) \* 30 +
7. ABS(g.price\_final - %s) \* 20 +
8. ABS(g.price\_original - %s) \* 20 +
9. ABS(g.discount - %s) \* 15 +
10. ABS(g.steam\_deck::int - %s::int) \* 2)/100 AS similarity\_score
11. FROM games g
12. WHERE g.app\_id <> %s
13. ORDER BY similarity\_score
14. LIMIT 25;"""

And by running this query, we can get the result like this:



The first row is the app\_id of the similar games, the second column is the positive ratio and the third column is the difference score. We take the 1 / difference score as weights, calculate the weighted average of positive ratio for the new game.

Then we can have this result:

The predicted positive ratio of this new game is 80.67178290619914

At last, I import the new game to neo4j and create a relation between this new game and its ‘similar’ games.

1. **def** create\_relationships(driver, game\_data, similarities):
2. with driver.session() as session:
3. session.run("""
4. CREATE (g:Game {app\_id: $app\_id, title: $title, win: $win, mac: $mac,
5. linux: $linux, user\_reviews: $user\_reviews, price\_final: $price\_final,
6. price\_original: $price\_original, discount: $discount, steam\_deck: $steam\_deck})
7. """, game\_data)
8. **for** \_, game\_id2, score **in** similarities:
9. session.run("""
10. MATCH (g1:Game {app\_id: $game\_id1}), (g2:Game {app\_id: $game\_id2})
11. MERGE (g1)-[r:SIMILAR]->(g2)
12. SET r.score = $score
13. """, {"game\_id1": game\_data['app\_id'], "game\_id2": game\_id2, "score": score})

