Recommendations with Neo4j and Python

Grant Beasley
Twitter - @grantbeasley
Blog - grant592.github.io

Senior Data Scientist at Sage

About Me

SaleCycle

Sage


HISISUS


New York Report

Content

- Context
- Recommendations
- Graph model
- Image embeddings
- Collaborative filtering
- Bringing it all together

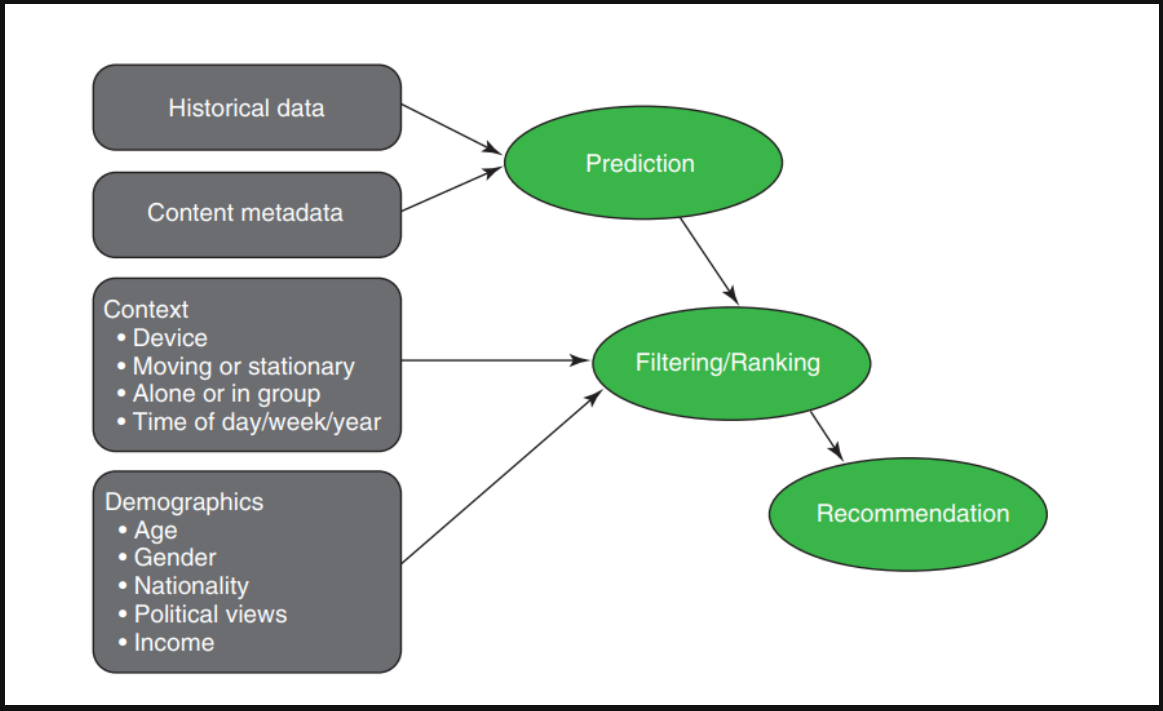
- Kim Falk - Practical Recommender Systems
- Ethan Rosenthal - blog

Recommendations

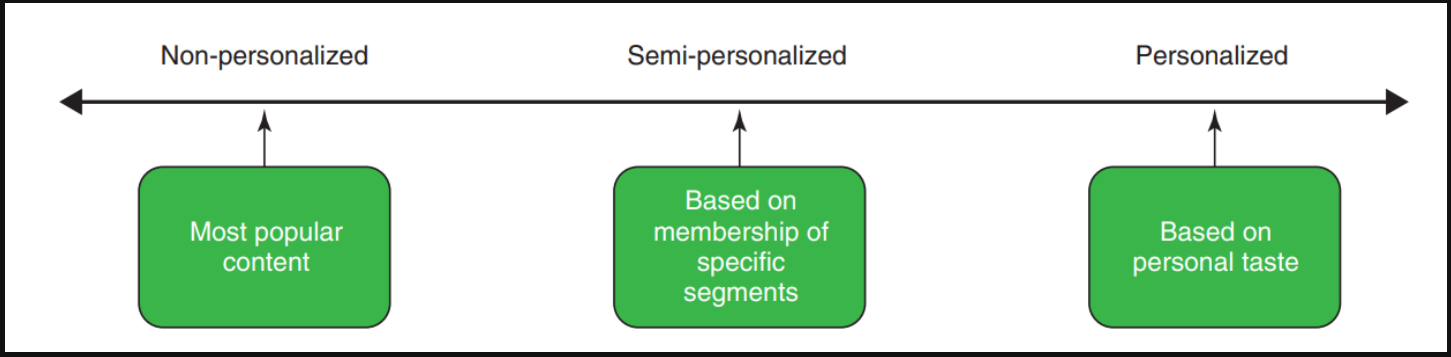
“A recommender system calculates and provides relevant content to the user based on knowledge of the user, content, and interactions between the user and the item.”

- The long tail
- Netflix

Taxonomy of Recommender System



Levels of Personalisation



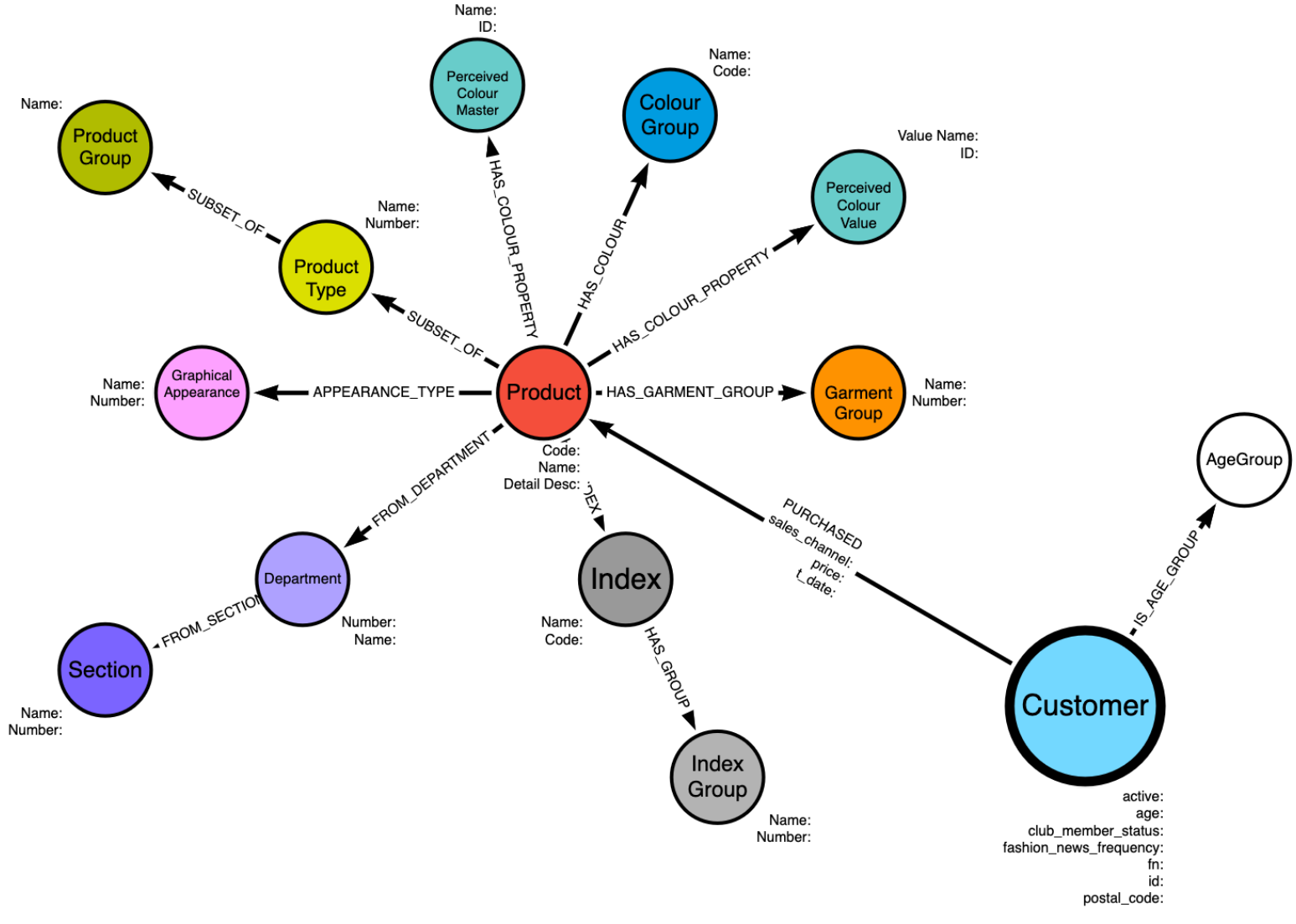
Other Considerations

- Tackling the cold start problem
- Grey sheep

Background and Why

- Kaggle Competition
- Combine different approaches
- Neo4j deep dive
- Safety Notice!

The Data Model



The Data Model

Loading with Cypher

Create indexes, load data, create relationships.

```
CREATE CONSTRAINT uniqueProduct IF NOT EXISTS
FOR (n:Product) REQUIRE (n.code) IS UNIQUE;

CREATE CONSTRAINT uniqueColourGroup IF NOT EXISTS
FOR (n:ColourGroup) REQUIRE (n.code) IS UNIQUE;

CREATE CONSTRAINT uniqueColourValue IF NOT EXISTS
FOR (n:PerceivedColourValue) REQUIRE (n.id) IS UNIQUE;

CREATE CONSTRAINT uniqueColourMaster IF NOT EXISTS
FOR (n:PerceivedColourMaster) REQUIRE (n.id) IS UNIQUE;

CREATE CONSTRAINT uniqueProductType IF NOT EXISTS
FOR (n:ProductType) REQUIRE (n.number) IS UNIQUE;

CREATE CONSTRAINT uniqueProductGroup IF NOT EXISTS
```

Loading with Cypher

Create indexes, load data, create relationships.

```
CREATE CONSTRAINT uniqueProduct IF NOT EXISTS
FOR (n:Product) REQUIRE (n.code) IS UNIQUE;

CREATE CONSTRAINT uniqueColourGroup IF NOT EXISTS
FOR (n:ColourGroup) REQUIRE (n.code) IS UNIQUE;

CREATE CONSTRAINT uniqueColourValue IF NOT EXISTS
FOR (n:PerceivedColourValue) REQUIRE (n.id) IS UNIQUE;

CREATE CONSTRAINT uniqueColourMaster IF NOT EXISTS
FOR (n:PerceivedColourMaster) REQUIRE (n.id) IS UNIQUE;

CREATE CONSTRAINT uniqueProductType IF NOT EXISTS
FOR (n:ProductType) REQUIRE (n.number) IS UNIQUE;

CREATE CONSTRAINT uniqueProductGroup IF NOT EXISTS
```

Loading with Cypher

```
LOAD CSV WITH HEADERS FROM 'file:///articles.csv' AS row
MERGE (ga:GraphicalAppearance {number: row.graphical_appearanc
ON CREATE SET
  ga.name = row.graphical_appearance_name
MERGE (dept:Department {number: row.department_no})
ON CREATE SET
  dept.name = row.department_name
MERGE (sect:Section {number: row.section_no})
ON CREATE SET
  sect.name = row.section_name
MERGE (indx:Index {code: row.index_code})
ON CREATE SET
  indx.name = row.index_name
MERGE (indxGrp:IndexGroup {number: row.index_group_no})
ON CREATE SET
  indxGrp.name = row.index_group_nam
```

Loading with Cypher

```
34 ON CREATE SET
35   prod.name = row.prod_name,
36   prod.description = row.detail_desc
37
38 WITH ga, dept, sect, indx, indxGrp, garmGrp, pcv, pcm, cg, pro
39 MERGE (prod)-[:HAS_COLOUR_VALUE]->(pcv)
40 MERGE (prod)-[:HAS_MASTER_COLOUR]->(pcm)
41 MERGE (prod)-[:HAS_COLOUR]->(cg)
42 MERGE (prod)-[:SUBSET_OF]->(prodType)
43 MERGE (prodType)-[:SUBSET_OF]->(prodGrp)
44 MERGE (prod)-[:HAS_APPEARANCE_TYPE]->(ga)
45 MERGE (prod)-[:FROM_DEPARTMENT]->(dept)
46 MERGE (dept)-[:FROM_SECTION]->(sect)

47 MERGE (prod)-[:HAS_INDEX]->(indx)
48 MERGE (indx)-[:HAS_GROUP]->(indxGrp)
```

Loading with Cypher

Customer data

```
CREATE CONSTRAINT uniqueCustomer IF NOT EXISTS FOR (n:Customer

LOAD CSV WITH HEADERS FROM 'file:///customers.csv' AS row
CALL {
  WITH row
  MERGE (c:Customer {id: row.customer_id})
  ON CREATE SET
    c.club_member_status =
      CASE
        WHEN  row.club_member_status IS NULL THEN 'NOT_ACTIVE'
        ELSE  row.club_member_status
      END,
    c.active =
      CASE
        WHEN row.Active IS NULL THEN 0
        ELSE row.Active
```

Loading with Cypher

Customer data

```
CREATE CONSTRAINT uniqueCustomer IF NOT EXISTS FOR (n:Customer

LOAD CSV WITH HEADERS FROM 'file:///customers.csv' AS row
CALL {
  WITH row
  MERGE (c:Customer {id: row.customer_id})
  ON CREATE SET
    c.club_member_status =
      CASE
        WHEN  row.club_member_status IS NULL THEN 'NOT_ACTIVE'
        ELSE  row.club_member_status
      END,
    c.active =
      CASE
        WHEN row.Active IS NULL THEN 0
        ELSE row.Active
```

Loading with Cypher

Customer data

```
24        CASE
25          WHEN row.fashion_news_frequency IS NULL THEN 'NONE'
26          ELSE row.fashion_news_frequency
27        END,
28      c.age =
29        CASE
30          WHEN row.age IS NULL THEN 100
31          ELSE row.age
32        END,
33      c.postal_code =
34        CASE
35          WHEN row.postal_code IS NULL THEN 'NONE'
36          ELSE row.postal_code
37        END
38
39  } IN TRANSACTIONS OF 1000 ROWS;
```

Loading with Cypher

Second pass of the data.

```
LOAD CSV WITH HEADERS FROM 'file:///transactions_train.csv' AS
CALL {
  WITH row
    MATCH (c:Customer {id: row.customer_id})
    MATCH (p:Product {code: row.article_id})
    MERGE (c)-[:PURCHASED {
        t_date: date(row.t_date),
        price: row.price,
        sales_channel: row.sales_channel_id}]->(p)
} IN TRANSACTIONS OF 1000 ROWS;
```

Bucketizing the data

Age categories

```
1 //
2 WITH [[16,25],[26,35],[36,45],[46,55],[56,65],[65,99],[100,100]
3 UNWIND ageGroups as ag
4 CREATE (:AgeGroup {lower: ag[0], upper: ag[1]});
5
6 // Create relationship between age group and customer
7 MATCH (c:Customer)
8 MATCH (a:AgeGroup) WHERE toInteger(c.age) >= a.lower AND toInte
9 CREATE (c)-[:IS_AGE_GROUP]->(a);
```

EDA

EDA

Label	Total Count

EDA

Label	Total Count
Customers	1,371,980

EDA

Label	Total Count
Customers	1,371,980
Product	105,542

EDA

Label	Total Count
Customers	1,371,980
Product	105,542
Purchases	28,813,419

EDA

EDA

Label	Average	Std Dev

EDA

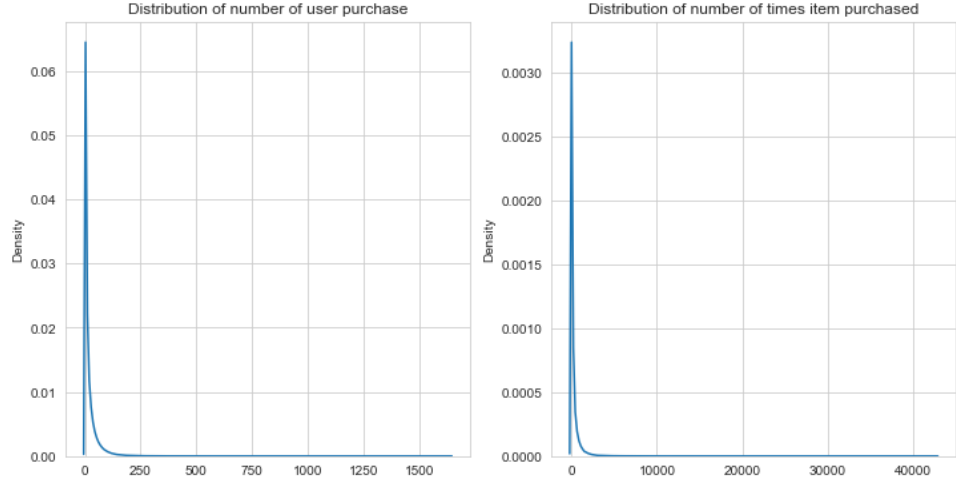
Label	Average	Std Dev
Customer Purchases	21.2	34.6

EDA

Label	Average	Std Dev
Customer Purchases	21.2	34.6
Product Purchased	275.6	696.9

EDA

Label	Average	Std Dev
Customer Purchases	21.2	34.6
Product Purchased	275.6	696.9



EDA

Most Popular Product Types

```
with driver.session() as session:
    query = """
    MATCH (p:Product)-[:SUBSET_OF]->(n:ProductType)
    RETURN n.name as product_type, COUNT(p) as count
    """
            data = list(session.run(query))

    df = pd.DataFrame([dict(record) for record in data])

    top_prods = df.sort_values('count', ascending=False).head(
```

EDA

Most Popular Product Types

```
with driver.session() as session:
    query = """
    MATCH (p:Product)-[:SUBSET_OF]->(n:ProductType)
    RETURN n.name as product_type, COUNT(p) as count
    """
            data = list(session.run(query))

    df = pd.DataFrame([dict(record) for record in data])

    top_prods = df.sort_values('count', ascending=False).head(
```

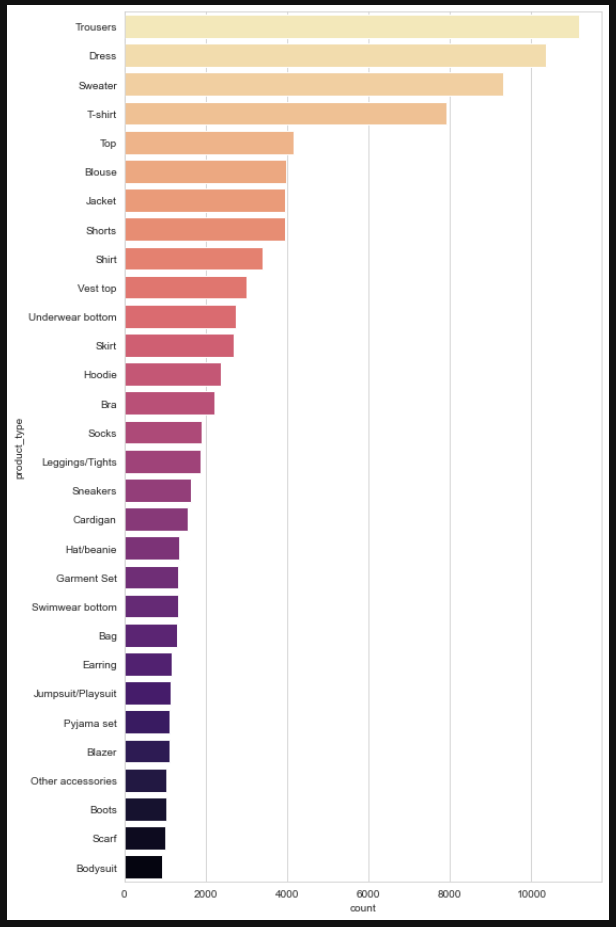
EDA

Most Popular Product Types

```
with driver.session() as session:
    query = """
    MATCH (p:Product)-[:SUBSET_OF]->(n:ProductType)
    RETURN n.name as product_type, COUNT(p) as count
    """
            data = list(session.run(query))

    df = pd.DataFrame([dict(record) for record in data])

    top_prods = df.sort_values('count', ascending=False).head(
```

EDA

Most Popular Product Types



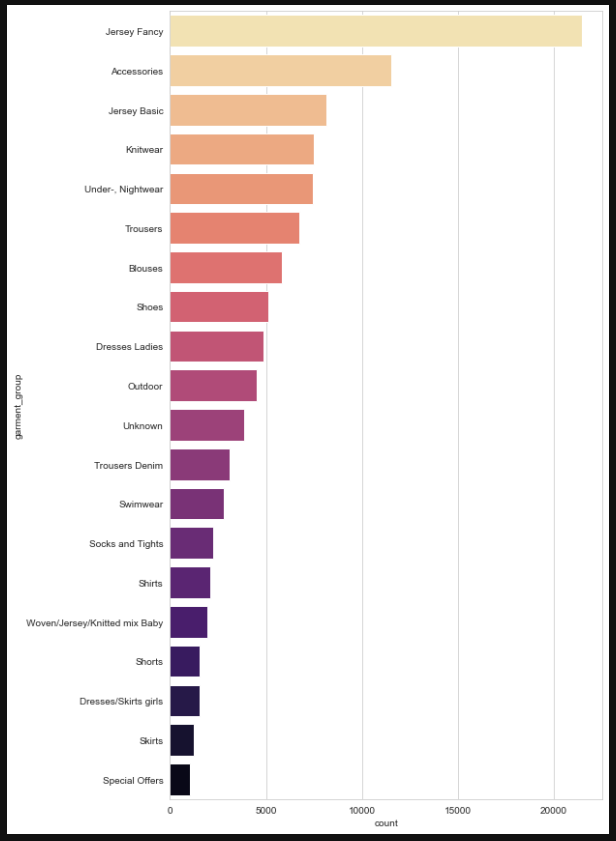
EDA

Most Popular Garment Types

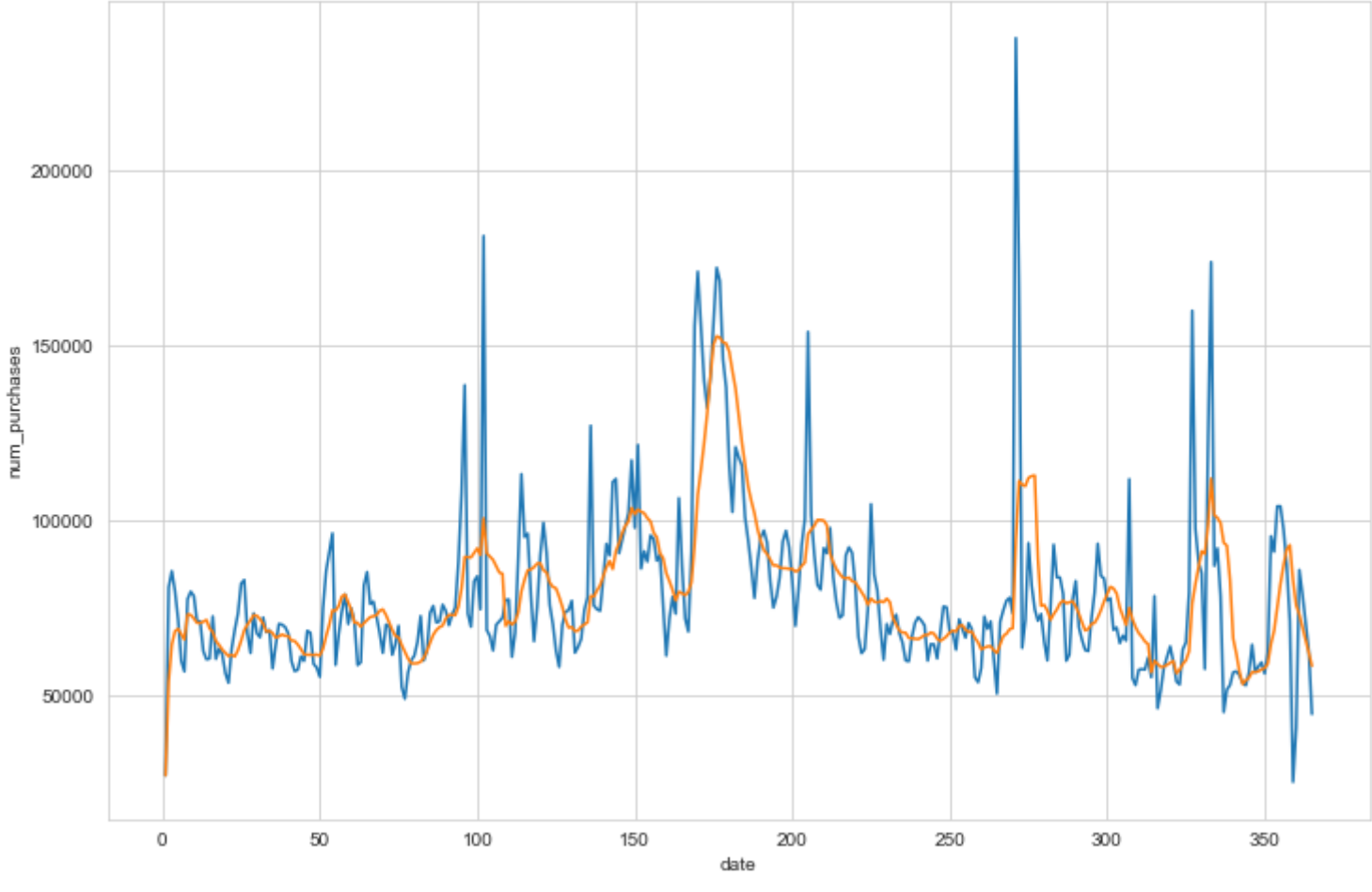
```
with driver.session() as session:
    query = """
    MATCH (p:Product)-[:HAS_GARMENT_GROUP]->(g:GarmentGroup)
    RETURN g.name as garment_group, COUNT(g) as count
    """
    data = list(session.run(query))
```

EDA

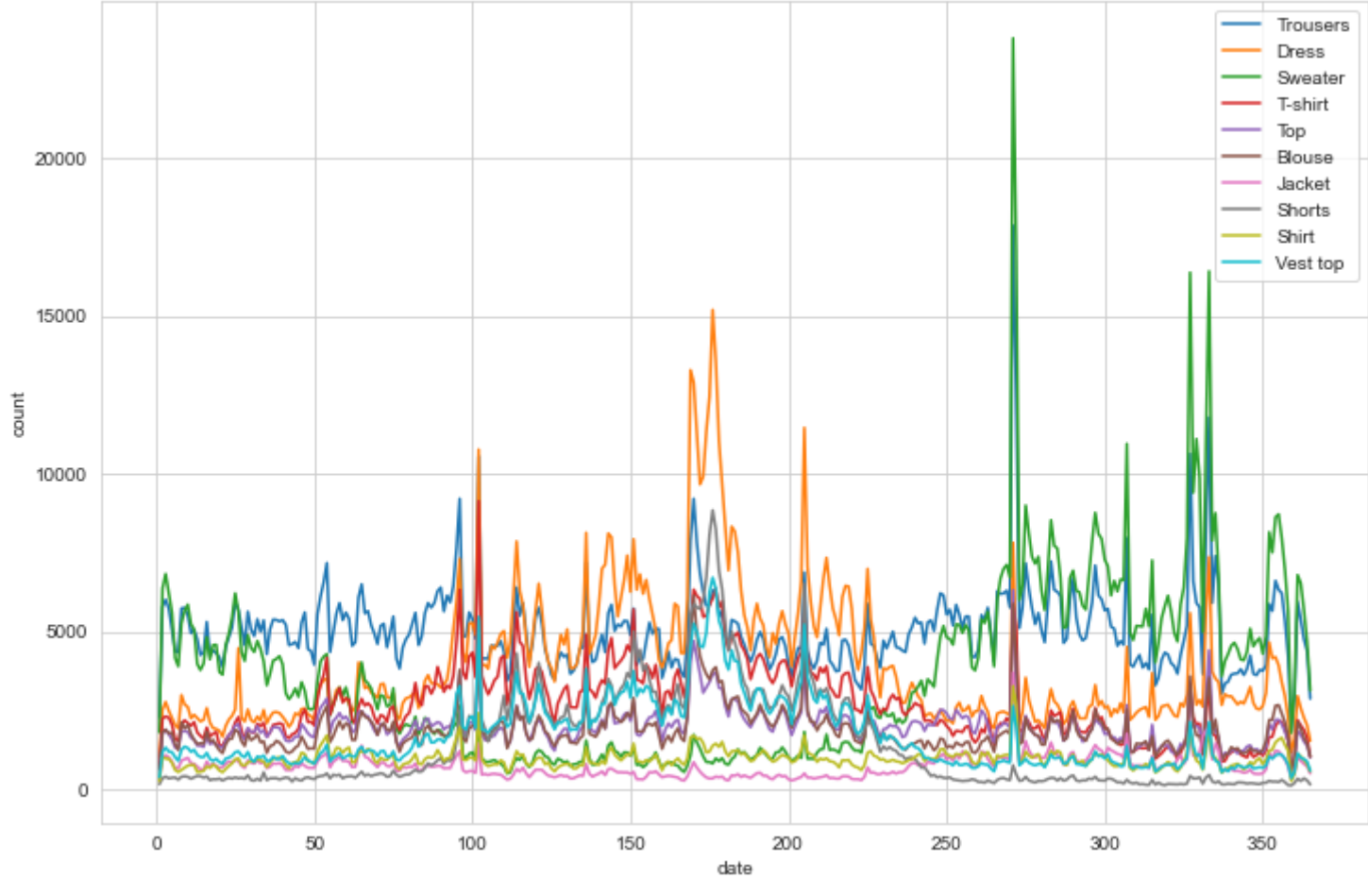
Most Popular Garment Types



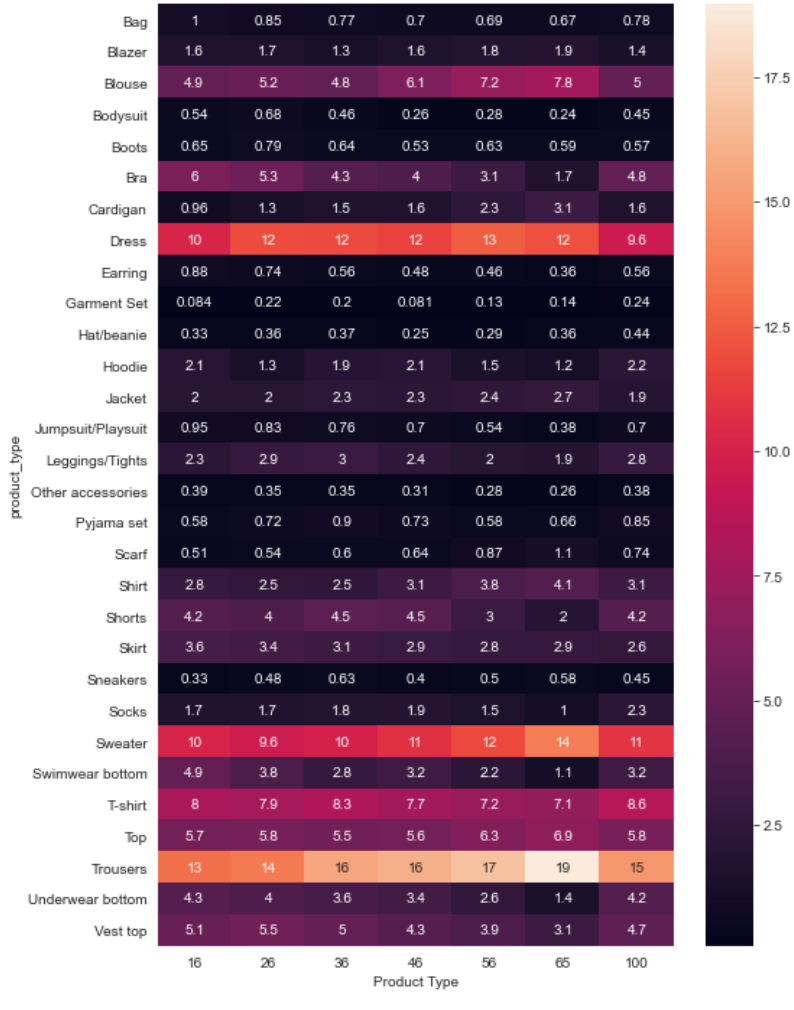
Seasonal Effects



Seasonal Effects



Age Effects


product_type
Bag 1 0.85 0.77 0.7 0.69 0.67 0.78
Blazer 1.6 1.7 1.3 1.6 1.8 1.9 1.4
Blouse 4.9 5.2 4.8 6.1 7.2 7.8 5
Bodysuit 0.54 0.68 0.46 0.26 0.28 0.24 0.45
Boots 0.65 0.79 0.64 0.53 0.63 0.59 0.57
Bra 6 5.3 4.3 4 3.1 1.7 4.8
Cardigan 0.96 1.3 1.5 1.6 2.3 3.1 1.6
Dress 10 12 12 12 13 12 9.6
Earring 0.88 0.74 0.56 0.48 0.46 0.36 0.56
Garment Set 0.084 0.22 0.2 0.081 0.13 0.14 0.24
Hat/beanie 0.33 0.36 0.37 0.25 0.29 0.36 0.44
Hoodie 2.1 1.3 1.9 2.1 1.5 1.2 2.2
Jacket 2 2 2.3 2.3 2.4 2.7 1.9
Jumpsuit/Playsuit 0.95 0.83 0.76 0.7 0.54 0.38 0.7
Leggings/Tights 2.3 2.9 3 2.4 2 1.9 2.8
Other accessories 0.39 0.35 0.35 0.31 0.28 0.26 0.38
Pyjama set 0.58 0.72 0.9 0.73 0.58 0.66 0.85
Scarf 0.51 0.54 0.6 0.64 0.87 1.1 0.74
Shirt 2.8 2.5 2.5 3.1 3.8 4.1 3.1
Shorts 4.2 4 4.5 4.5 3 2 4.2
Skirt 3.6 3.4 3.1 2.9 2.8 2.9 2.6
Sneakers 0.33 0.48 0.63 0.4 0.5 0.58 0.45
Socks 1.7 1.7 1.8 1.9 1.5 1 2.3
Sweater 10 9.6 10 11 12 14 11
Swimwear bottom 4.9 3.8 2.8 3.2 2.2 1.1 3.2
T-shirt 8 7.9 8.3 7.7 7.2 7.1 8.6
Top 5.7 5.8 5.5 5.6 6.3 6.9 5.8
Trousers 13 14 16 16 17 19 15
Underwear bottom 4.3 4 3.6 3.4 2.6 1.4 4.2
Vest top 5.1 5.5 5 4.3 3.9 3.1 4.7
16 26 36 46 56 65 100
Product Type
17.5
15.0
12.5
10.0
7.5
5.0
2.5

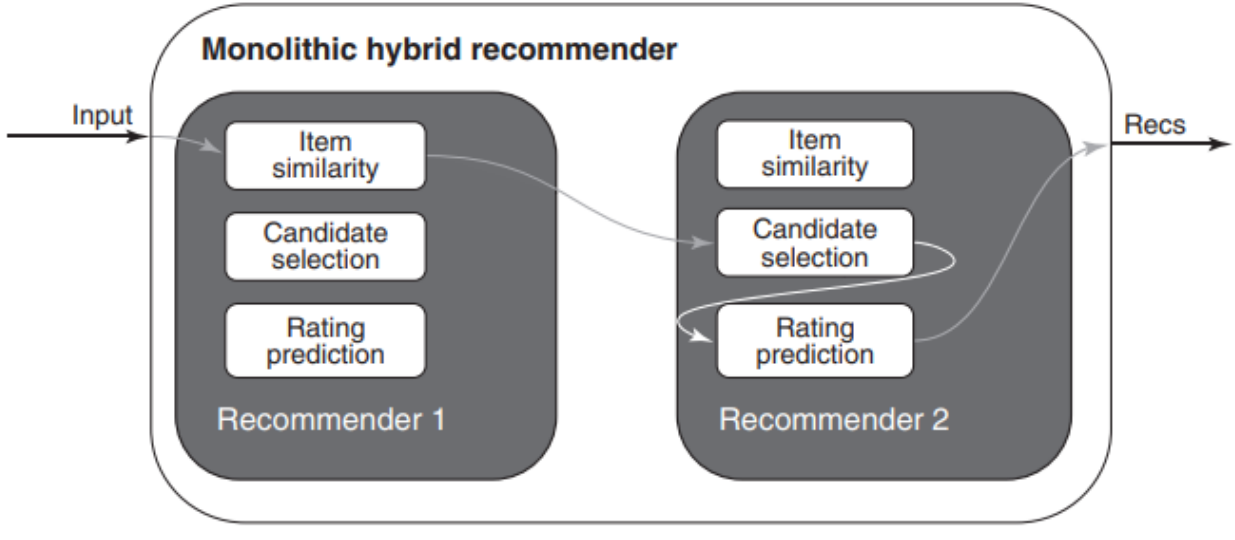

Age Effects
s they just don't make 'em like they use
No! Nobody ever made them like this!
uftgifs | tumblr


Exploration over...let's get to work

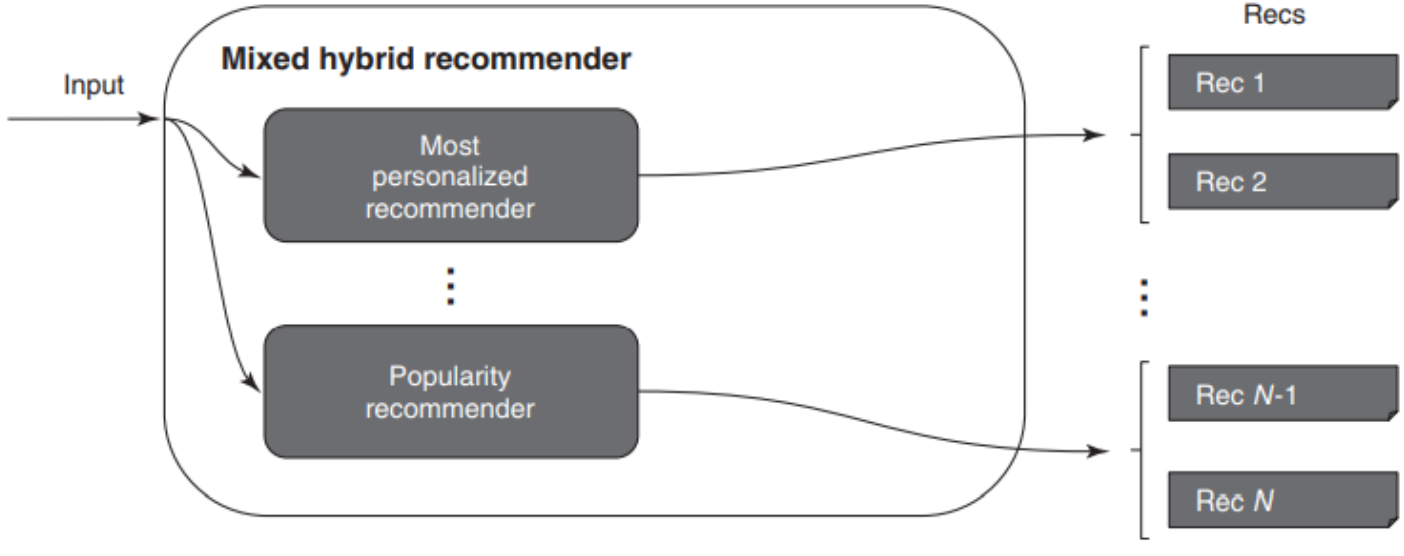
Considerations

- How to make best use of graph database?
- How to approach recommendations

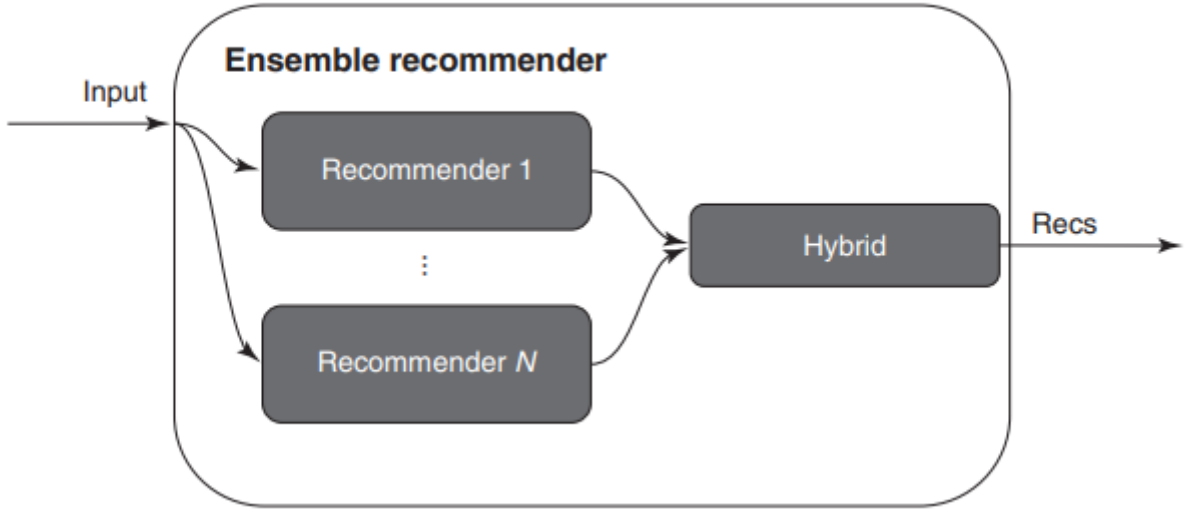
Recommender System



Recommender System

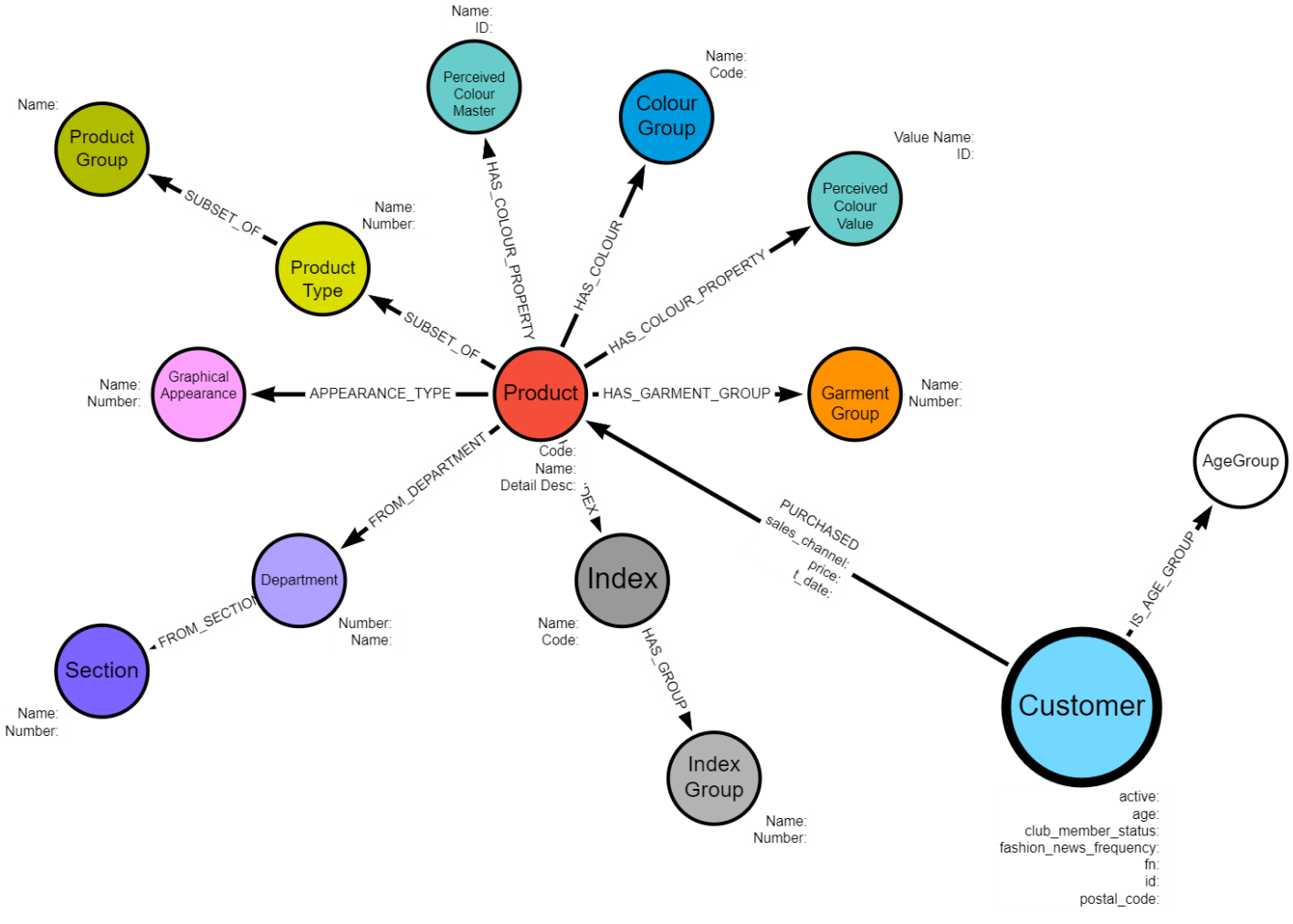


Recommender System



Neo4j Fabric

The initial data model

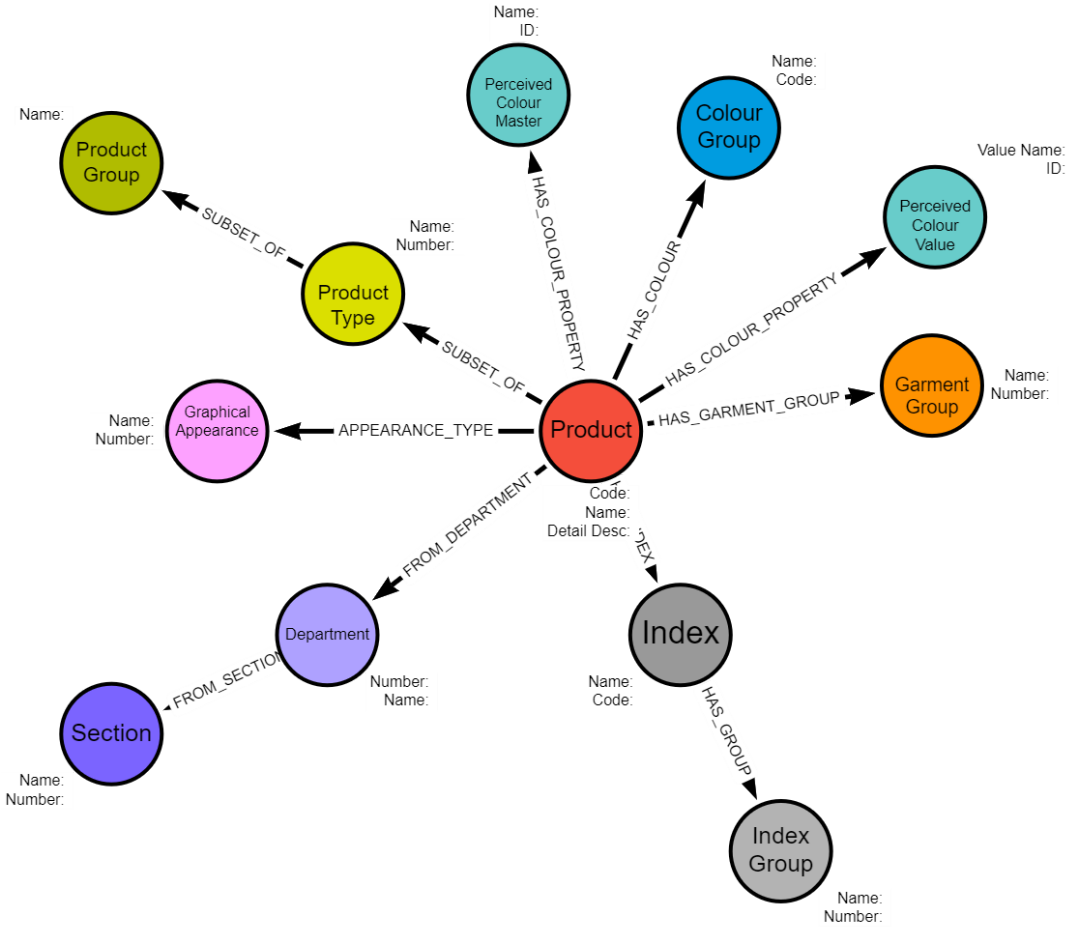


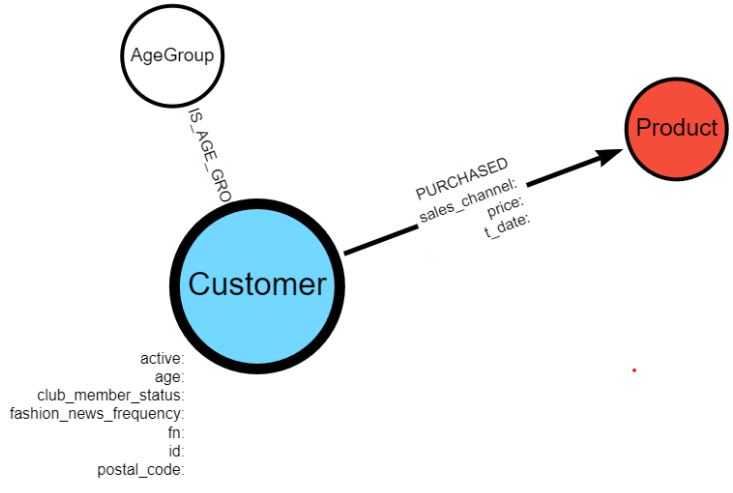
image_embedding:

Product

Neo4j Fabric

What it should have looked liked...





Neo4j Fabric

Fabric Setup

```
#Fabric config
fabric.database.name=fabric

fabric.graph.0.uri=neo4j://localhost:7687
fabric.graph.0.database=neo4j
fabric.graph.0.name=neo4j

fabric.graph.1.uri=neo4j://localhost:7687
fabric.graph.1.database=products
fabric.graph.1.name=products
```

Neo4j Fabric

Fabric Setup

```
# Load the second DB
:use system;
create database products;
use products:
CREATE CONSTRAINT uniqueProduct IF NOT EXISTS FOR (n:Product) R

LOAD CSV WITH HEADERS FROM 'file:///articles.csv' AS row
MERGE (prod:Product {code: row.article_id})
```

Warning!

Explore vs exploit

Image Embeddings and Transfer Learning

"Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks."

Image Embeddings and Transfer Learning

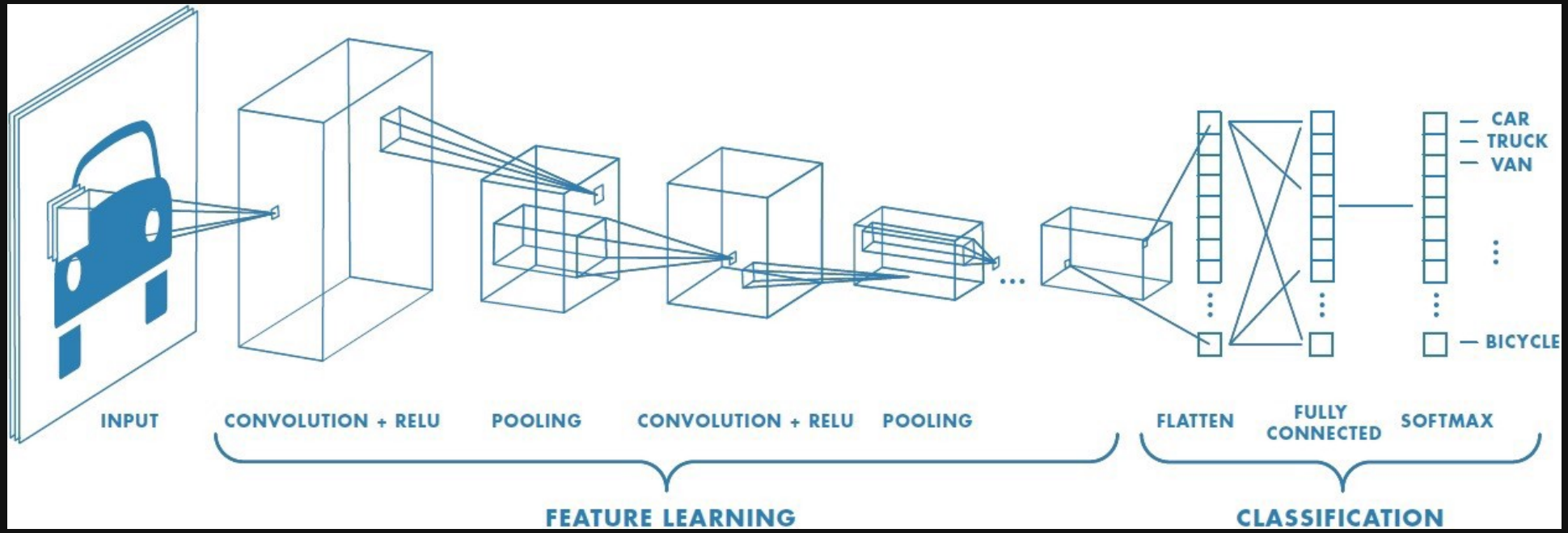
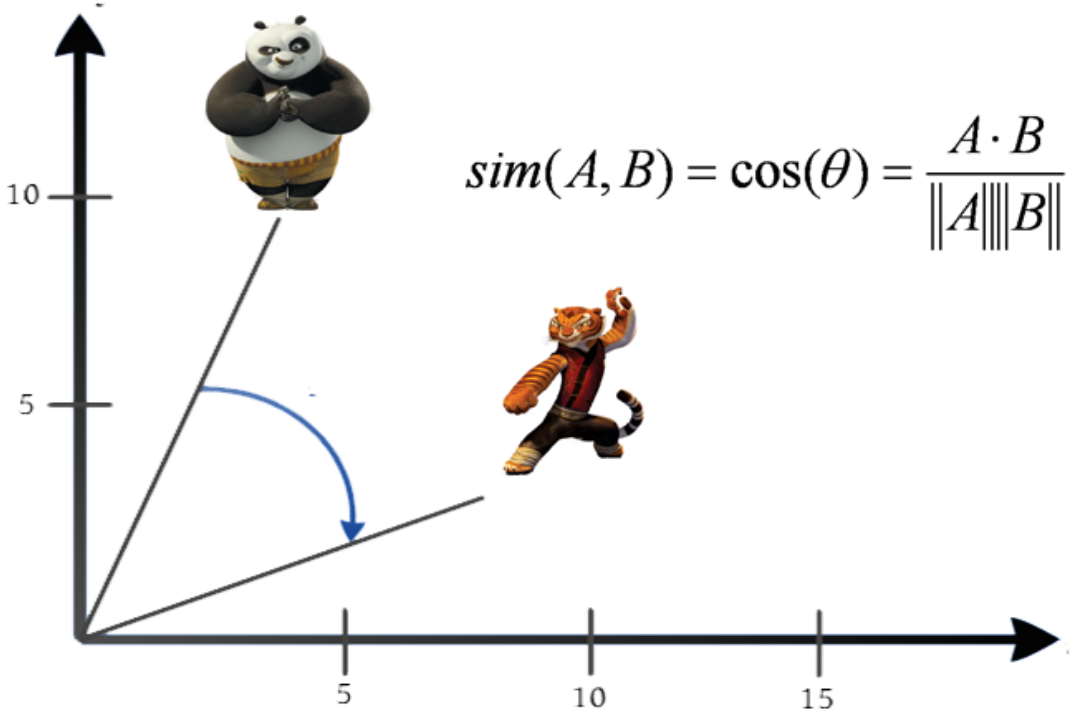


Image Embeddings and Transfer Learning

Cosine Similarity



$$sim(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Image Embeddings

Image Embedding Class

```
ImageEmbedder():
    def __init__(self):
        self._IMAGE_TARGET_SIZE = (224,224)

        # Create an instance of a pretrained resnet model
        # include_top to include the 2048x1 embedding layer

        pre_trained_model = resnet_v2.ResNet50V2(
          input_shape = (224, 224, 3),
          weights = 'imagenet',
          include_top=True
        )

      # Create a model that outputs only the second to last la
      self.embedding_model = Model(
          inputs=pre_trained_model.input,
```

Image Embeddings

Image Embedding Class

```
 3         self._IMAGE_TARGET_SIZE = (224,224)
 4
 5         # Create an instance of a pretrained resnet model
 6         # include_top to include the 2048x1 embedding layer
 7
 8         pre_trained_model = resnet_v2.ResNet50V2(
 9           input_shape = (224, 224, 3),
10           weights = 'imagenet',
11           include_top=True
12         )
13
14       # Create a model that outputs only the second to last la
15       self.embedding_model = Model(
16           inputs=pre_trained_model.input,
17           outputs=pre_trained_model.get_layer('avg_pool').outp
18         )
```

Image Embeddings

Image Embedding Class

```
10            weights = 'imagenet',
11            include_top=True
12          )
13
14        # Create a model that outputs only the second to last la
15        self.embedding_model = Model(
16            inputs=pre_trained_model.input,
17            outputs=pre_trained_model.get_layer('avg_pool').outp
18          )
19
20        def create_embedding(self, image_path):
21          img = image.load_img(
22          image_path,
23          target_size=self._IMAGE_TARGET_SIZE
24          )
```

Image Embeddings

Image Embedding Class

```
13
14        # Create a model that outputs only the second to last la
15        self.embedding_model = Model(
16            inputs=pre_trained_model.input,
17            outputs=pre_trained_model.get_layer('avg_pool').outp
18          )
19
20        def create_embedding(self, image_path):
21          img = image.load_img(
22          image_path,
23          target_size=self._IMAGE_TARGET_SIZE
24          )
25
26          # Convert the image to an array
27          img_vec = image.img_to_array(img)
```

Image Embeddings

Image Embedding Class

```
22            image_path,
23            target_size=self._IMAGE_TARGET_SIZE
24            )
25
26            # Convert the image to an array
27            img_vec = image.img_to_array(img)
28            # The model expects batches of images in the shape
29            # (n,224,224,3), so add an extra dimension
30            img_vec = np.expand_dims(img_vec, axis=0)
31
32            # resnet_v2 has a built in function to preprocess
33            # images for the resnet model
34            img_vec = resnet_v2.preprocess_input(img_vec)
35
36            # Return the embedding from the function
37            return self.embedding_model.predict(img_vec)
```

Image Embeddings

Neo4j Import Class

```
import logging
import sys
from neo4j import GraphDatabase
from neo4j.exceptions import ServiceUnavailable

class NeoProdEmbeddings:
  def __init__(self, uri, user, password):
    self.driver = GraphDatabase.driver(uri, auth=(user, passw

  def close(self):
  # Don't forget to close the driver connection when you are
    self.driver.close()

  @staticmethod
  def enable_log(level, output_stream):
    handler = logging.StreamHandler(output_stream)
```

Image Embeddings

Neo4j Import Class

```
      handler = logging.StreamHandler(output_stream)
      handler.setLevel(level)
      logging.getLogger("neo4j").addHandler(handler)
      logging.getLogger("neo4j").setLevel(level)

    def add_prod_embedding(self, prod_id, embedding):
      with self.driver.session(database='products') as session:
        # Write transactions allow the driver to handle retries
        result = session.write_transaction(
          self._add_and_return_embedding, prod_id, embedding
  )

for row in result:
  print(f"Created embedding for product ID {prod_id}")

    @staticmethod
```

Image Embeddings

Neo4j Import Class

```
for row in result:
  print(f"Created embedding for product ID {prod_id}")

   @staticmethod
   def _add_and_return_embedding(tx, prod_id, embedding):
     query = (
              "MERGE (prod:Product { code: $prod_id }) "
              "ON MATCH set prod.embedding = $embedding "
              "RETURN prod.code as prod_code, prod.embedding[0
)

     result = tx.run(query, prod_id=prod_id,embedding=embeddin

     try:
       return [
  {
```

Image Embeddings

Neo4j Import Class

```
import logging
import sys
from neo4j import GraphDatabase
from neo4j.exceptions import ServiceUnavailable

class NeoProdEmbeddings:
  def __init__(self, uri, user, password):
    self.driver = GraphDatabase.driver(uri, auth=(user, passw

  def close(self):
  # Don't forget to close the driver connection when you are
    self.driver.close()

  @staticmethod
  def enable_log(level, output_stream):
    handler = logging.StreamHandler(output_stream)
```

Image Embeddings

Running the code

```
from embed_functions import image_embedder, neo_helper
from pathlib import Path

if __name__=='__main__':
  uri = "bolt://localhost:7687"
  user = 'neo4j'
  password = 'password'

# Instatiate each of the classes we made
  img_emb = image_embedder.ImageEmbedder()
  neo_emb = neo_helper.NeoProdEmbeddings(uri, user, password)

  IMAGE_DIR = '/Users/grantbeasley/Downloads/images/'
  images = Path(IMAGE_DIR)

  # Find all of the image files
```

Image Embeddings

Running the code

```
 3
 4 if __name__=='__main__':
 5   uri = "bolt://localhost:7687"
 6   user = 'neo4j'
 7   password = 'password'
 8
 9 # Instatiate each of the classes we made
10   img_emb = image_embedder.ImageEmbedder()
11   neo_emb = neo_helper.NeoProdEmbeddings(uri, user, password)
12
13   IMAGE_DIR = '/Users/grantbeasley/Downloads/images/'
14   images = Path(IMAGE_DIR)
15
16   # Find all of the image files
17
18   all_files = [
```

Image Embeddings

Running the code

```
16  # Find all of the image files
17
18  all_files = [
19    image for folder in images.iterdir()
20    if not folder.name == '.DS_Store'
21    for image in folder.iterdir()
22    if image.suffix == '.jpg'
23  ]
24
25  # Iterate through each image an upload to the database
26  for file in all_files:
27    prod_id = file.name.replace('.jpg', '')
28    print(prod_id)
29    embedding = img_emb.create_embedding(file)
30    neo_emb.add_prod_embedding(prod_id, embedding[0].tolist())
```

Finding similar items by image

```
// Use appropriate database
:use products;

// Create graph projection with node properties
CALL gds.graph.project(
  'knnGraph', {
    Product: {
      label: 'Product',
properties:'embedding'
    }
}, '*');

// Create the top 10 nearest neighbours based on the image em
CALL gds.beta.knn.write(
  'knnGraph', {
    writeRelationshipType: 'SIMILAR',
```

Finding similar items by image

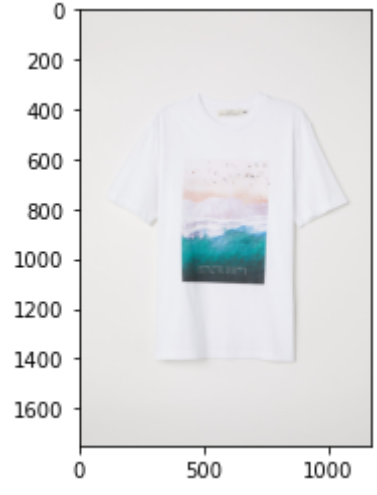
```
// Use appropriate database
:use products;

// Create graph projection with node properties
CALL gds.graph.project(
  'knnGraph', {
    Product: {
      label: 'Product',
properties:'embedding'
    }
}, '*');

// Create the top 10 nearest neighbours based on the image em
CALL gds.beta.knn.write(
  'knnGraph', {
    writeRelationshipType: 'SIMILAR',
```

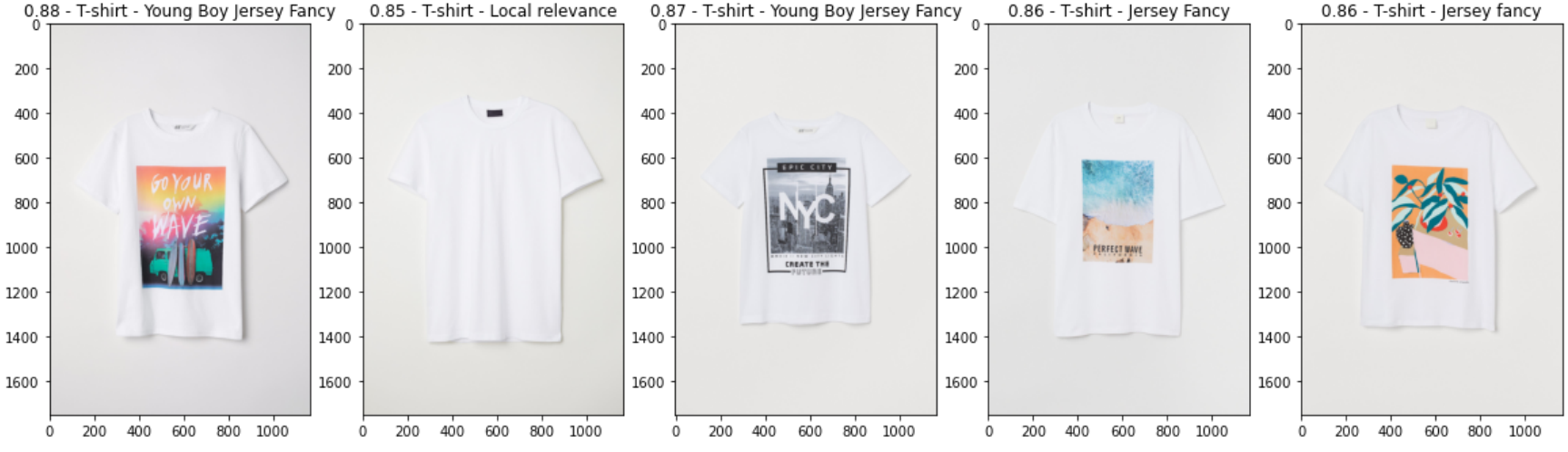
Finding similar items by image

```
    }
}, '*');

// Create the top 10 nearest neighbours based on the image em
CALL gds.beta.knn.write(
  'knnGraph', {
    writeRelationshipType: 'SIMILAR',
    writeProperty: 'score',
    topK: 10,
    randomSeed: 42,
    concurrency: 1,
    nodeWeightProperty: 'embedding'
  }
)

YIELD nodesCompared, relationshipsWritten;
```

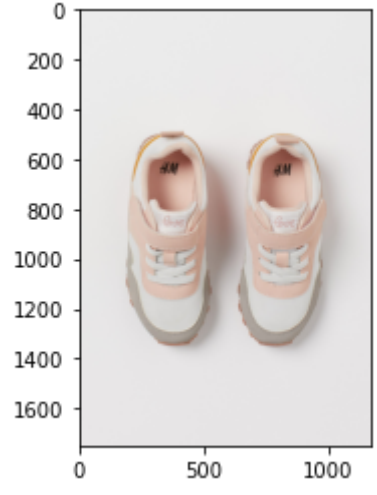
Randomly sampling data for quick validation

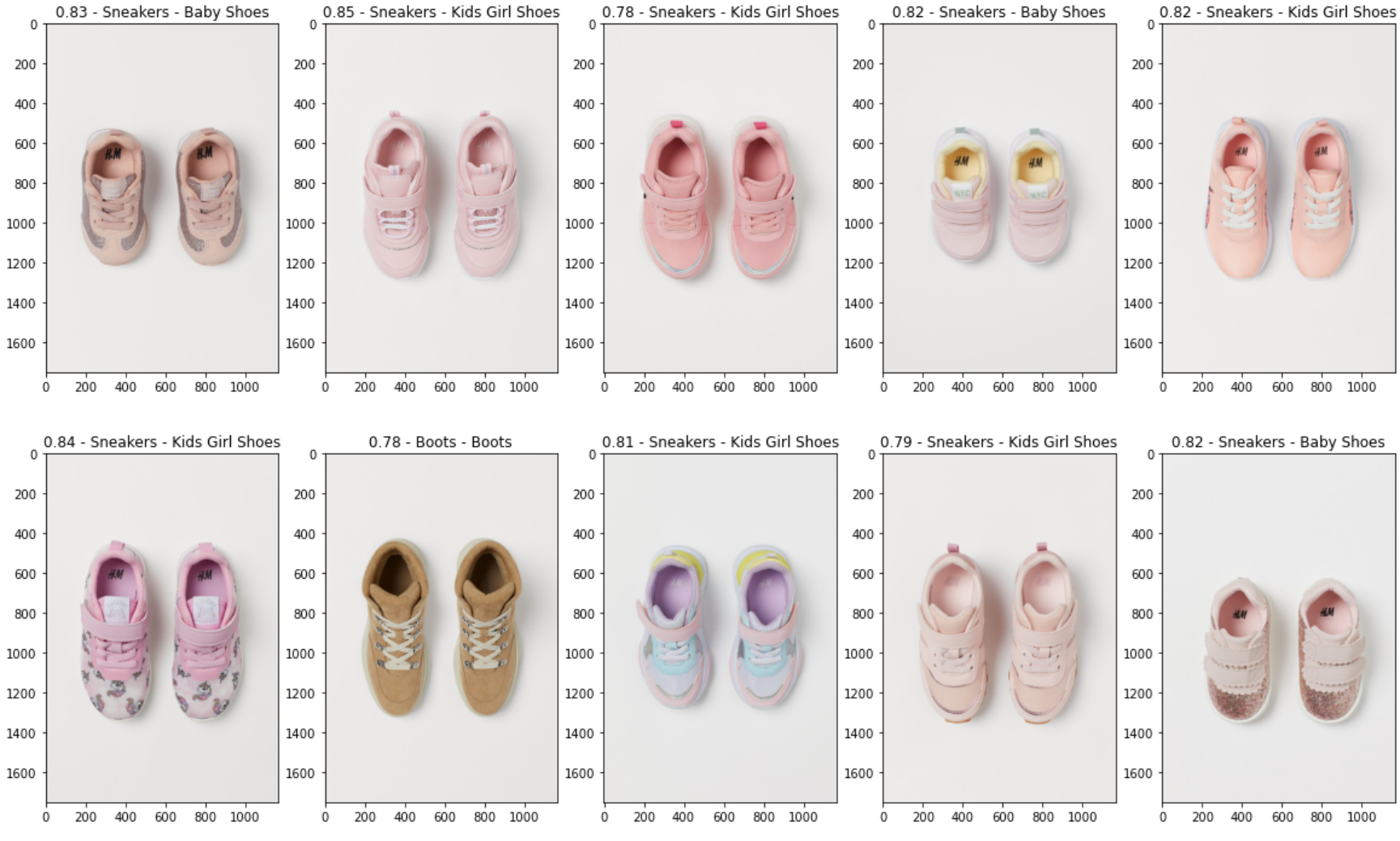
```
CALL {
  USE fabric.products
  MATCH (p:Product {code: $prod_code})-[s:SIMILAR]->(p2:Produc
  RETURN p2.code as prod_code, s.score as sim_score
}
CALL {
  USE fabric.neo4j
  WITH prod_code
  MATCH (d:Department)<-[:FROM_DEPARTMENT]-(p:Product {code:pr
  RETURN type.name as prod_type, d.name as department_name
}

RETURN prod_code, sim_score, prod_type, department_name
```


0
200
400
600
800
1000
1200
1400
1600
0
500
1000


0.88 - T-shirt - Young Boy Jersey Fancy
0.85 - T-shirt - Local relevance
0.87 - T-shirt - Young Boy Jersey Fancy
0.86 - T-shirt - Jersey Fancy
0.86 - T-shirt - Jersey fancy
GO YOUR OWN WAVE
EPIC CITY
NYC
CREATE THE
PERFECT WAVE


0.85 - T-shirt - Young Boy Jersey Fancy
0.87 - T-shirt - Kids Boy Jersey Fancy
0.86 - T-shirt - Jersey Fancy
0.85 - Top - Jersey fancy
0.86 - T-shirt - Jersey Fancy
VACATION MODE ON
L.A.
MI CASA ES SU CASA


0
200
400
600
800
1000
1200
1400
1600
0
500
1000


0.83 - Sneakers - Baby Shoes
0.85 - Sneakers - Kids Girl Shoes
0.78 - Sneakers - Kids Girl Shoes
0.82 - Sneakers - Baby Shoes
0.82 - Sneakers - Kids Girl Shoes
0.84 - Sneakers - Kids Girl Shoes
0.78 - Boots - Boots
0.81 - Sneakers - Kids Girl Shoes
0.79 - Sneakers - Kids Girl Shoes
0.82 - Sneakers - Baby Shoes


0
200
400
600
800
1000
1200
1400
1600
0
500
1000


0.80 - Sweater - Kids Boy Knitwear
0.81 - Shirt - Shirt
0.80 - Sweater - Kids Girl Knitwear
0.84 - Top - Kids Boy Jersey Basic
0.80 - Sweater - Kids Boy Jersey Basic
0.83 - Sweater - Kids Boy Jersey Fancy
0.84 - Sweater - Kids Boy Knitwear
0.81 - Sweater - Kids Boy Jersey Fancy
0.81 - Polo shirt - Knitwear
0.85 - Sweater - Kids Boy Jersey Basic

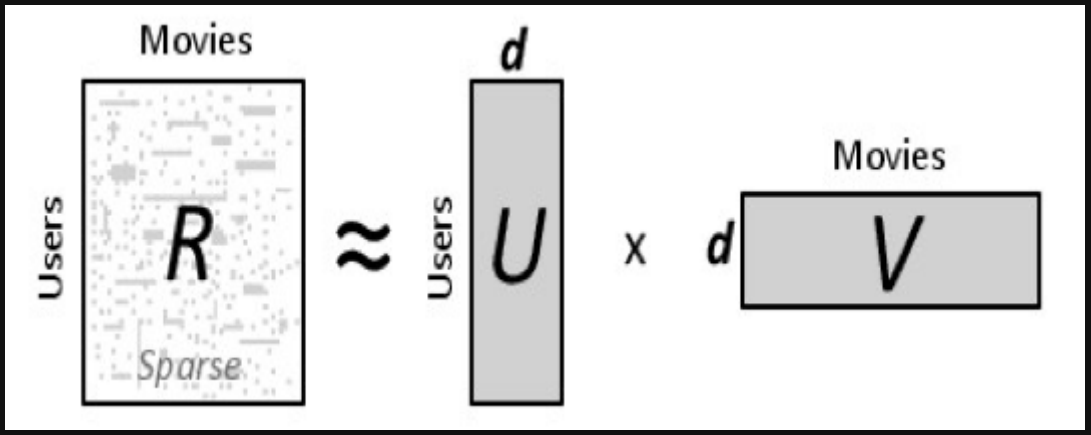
Collaborative Filtering and Implicit Feedback

Implicit Feedback

- No negative feedback
- Implicit feedback is noisy
- Explicit feedback indicates preference, implicit feedback indicates confidence
- Need to use different measures

Alternating Least Squares

Python package - implicit



Considerations

- When are we recommending for?
- Sparsity of our dataset

Considerations

- When are we recommending for?
- Sparsity of our dataset

Num of users	1,362,281

Considerations

- When are we recommending for?
- Sparsity of our dataset

Num of users	1,362,281
Number of items	104,547

Considerations

- When are we recommending for?
- Sparsity of our dataset

Num of users	1,362,281
Number of items	104,547
Sparsity	0.022%

Dealing With Sparsity

“While implicit recommendations excel where data is sparse, it can often be helpful to make the interactions matrix a little more dense. We limited our data collection to models that had at least 5 likes. However, it may not be the case that every user has liked at least 5 models. Let’s go ahead and knock out users that liked fewer than 5 models. This could possibly mean that some models end up with fewer than 5 likes once these users are knocked out, so we will have to alternate back and forth knocking users and models out until things stabilize.”

Decreasing the sparsity of the dataset

```
1   def threshold_likes(df, user_min, article_min):
2     n_users = df['customer_id'].unique().shape[0]
3     n_items = df['article_id'].unique().shape[0]
4     sparsity = float(df.shape[0]) / float(n_users*n_items) * 1
5
6     done = False
7     while not done:
8       starting_shape = df.shape[0]
9       article_counts = df.groupby('customer_id')['article_id']
10      df = df[~df['customer_id'].isin(article_counts[article_c
11      user_counts = df.groupby('article_id')['customer_id'].co
12      df = df[~df['article_id'].isin(user_counts[user_counts <
13      ending_shape = df.shape[0]
14      if starting_shape == ending_shape:
15        done = True
16
```

Decreasing the sparsity of the dataset

```
sparsity = float(df.shape[0]) / float(n_users*n_items) * 1

done = False
while not done:
  starting_shape = df.shape[0]
  article_counts = df.groupby('customer_id')['article_id']
  df = df[~df['customer_id'].isin(article_counts[article_c
  user_counts = df.groupby('article_id')['customer_id'].co
  df = df[~df['article_id'].isin(user_counts[user_counts <
  ending_shape = df.shape[0]
  if starting_shape == ending_shape:
    done = True

assert(df.groupby('customer_id')['article_id'].count().min
assert(df.groupby('article_id')['customer_id'].count().min
```

Decreasing the sparsity of the dataset

```
18      assert(df.groupby('article_id')['customer_id'].count().min
19
20    n_users = df['customer_id'].unique().shape[0]
21    n_items = df['article_id'].unique().shape[0]
22    sparsity = float(df.shape[0]) / float(n_users*n_items) * 100
23
24    return df
25
26  >>> Starting likes info
27  >>> Number of users: 1362281
28  >>> Number of items: 104547
29  >>> Sparsity: 0.022%
30  >>> Ending likes info
31  >>> Number of users: 925154
32  >>> Number of items: 91511
33  >>> Sparsity: 0.036%
```

Decreasing the sparsity of the dataset

```
1   def threshold_likes(df, user_min, article_min):
2     n_users = df['customer_id'].unique().shape[0]
3     n_items = df['article_id'].unique().shape[0]
4     sparsity = float(df.shape[0]) / float(n_users*n_items) * 1
5
6     done = False
7     while not done:
8       starting_shape = df.shape[0]
9       article_counts = df.groupby('customer_id')['article_id']
10      df = df[~df['customer_id'].isin(article_counts[article_c
11      user_counts = df.groupby('article_id')['customer_id'].co
12      df = df[~df['article_id'].isin(user_counts[user_counts <
13      ending_shape = df.shape[0]
14      if starting_shape == ending_shape:
15        done = True
16
```

We lose 32% of users and 12% of items!

Splitting the data

- Train test split
- Implications for number of purchases
- Precision @ k

Training the Model

Split the data and train on the training data.

```
als = implicit.als.AlternatingLeastSquares()

als.fit(train_data)
```

Eyeballing the Recommendations

$$P = \frac{\#\text{ of our recommendations that are relevant}}{\#\text{ of items we recommended}}$$

Precision

$$P = \frac{\#\text{ of our recommendations that are relevant}}{\#\text{ of items we recommended}}$$

Precision

Test - Precision @ 5 3.3%

$$P = \frac{\#\text{ of our recommendations that are relevant}}{\#\text{ of items we recommended}}$$

Precision

Test - Precision @ 5	3.3%
Train - Precision @ 5	49.6%

Purchase Confidence - 2 Possible Approaches

Assume most recent purchases are most relevant and assign more weight

```
def assign_confidence_recent(date):
  if date >= dt.date(2020,3,20):
    return 5
  elif (date >= dt.date(2019,9,20)) & (date < dt.date(2020,3,20))
    return 3
  else:
    return 1
```

Purchase Confidence - 2 Possible Approaches

We could assume that recent purchases are most relevant, and that purchases within the same season may be related (i.e. user preferences during autumn/winter may show different correlations to spring/summer).

```
def assign_confidence(date):
  if date >= dt.date(2020,6,20):
    return 5
  elif date.month in (7,8,9):
    return 3
  else:
    return 1
```

Results

Results

Binary Confidence	Confidence 1	Confidence 2

Results

	Binary Confidence	Confidence 1	Confidence 2
Train	49.6%	48.7%	48.7%

Results

	Binary Confidence	Confidence 1	Confidence 2
Train	49.6%	48.7%	48.7%
Test	3.3%	10.5%	10.4%

Bringing it all together

What we currently have:

- A graph model
- Image embeddings and similarities
- A matrix factorization based recommendation system

Challenges

- Image Embeddings - powerful but has limitations
- Matrix factorization method - decreasing sparsity of the dataset also introduces limitations

How we'll combine the various methods

How we'll combine the various methods

1. For each customer, find all purchases and retrieve most similar products based on images

How we'll combine the various methods

1. For each customer, find all purchases and retrieve most similar products based on images
2. Use the graph model and Cypher to filter list and ensure it only contains products from a department the customer has previously purchased from

How we'll combine the various methods

1. For each customer, find all purchases and retrieve most similar products based on images
2. Use the graph model and Cypher to filter list and ensure it only contains products from a department the customer has previously purchased from
3. Use the ALS model to rank all these items and take the top 12

How we'll combine the various methods

1. For each customer, find all purchases and retrieve most similar products based on images
2. Use the graph model and Cypher to filter list and ensure it only contains products from a department the customer has previously purchased from
3. Use the ALS model to rank all these items and take the top 12
4. For those people with less than 12 recommended items, take a random sample from the items from the Cypher query

How we'll combine the various methods

1. For each customer, find all purchases and retrieve most similar products based on images
2. Use the graph model and Cypher to filter list and ensure it only contains products from a department the customer has previously purchased from
3. Use the ALS model to rank all these items and take the top 12
4. For those people with less than 12 recommended items, take a random sample from the items from the Cypher query
5. For those still without 12 recommendations, use the graph model to return up to 12 most popular products purchased by other users who also purchased products by the current customer

Creating the Sparse Matrix

```
# Create mappings for sparse matrix
all_product_to_idx = {}
all_idx_to_product = {}
for (idx, prod_id) in enumerate(products):
    all_product_to_idx[prod_id] = idx
    all_idx_to_product[idx] = prod_id

all_customer_to_idx = {}
all_idx_to_customer = {}
for (idx, cust_id) in enumerate(customers):
    all_customer_to_idx[cust_id] = idx
    all_idx_to_customer[idx] = cust_id

# Create lil matrix to incrementally build sparse matrix of re
n_users = len(customers)
n_products = len(products)
```

The Recommendation Code

```
CALL {
// Return all purchases for a single customer
  USE fabric.neo4j
  MATCH (c:Customer)-[pur:PURCHASED]->(p:Product)
  WHERE  c.id = $customer_id
  RETURN p.code as prod_code
}
CALL {
// For each of those purchases, return the 10 most similar
// by image embedding
  USE fabric.products
  WITH prod_code
  MATCH (p1:Product)-[:SIMILAR]->(rec:Product)
  WHERE p1.code = prod_code
  RETURN rec.code as rec_code
}
```

The Recommendation Code

```
CALL {
// Return all purchases for a single customer
  USE fabric.neo4j
  MATCH (c:Customer)-[pur:PURCHASED]->(p:Product)
  WHERE  c.id = $customer_id
  RETURN p.code as prod_code
}
CALL {
// For each of those purchases, return the 10 most similar
// by image embedding
  USE fabric.products
  WITH prod_code
  MATCH (p1:Product)-[:SIMILAR]->(rec:Product)
  WHERE p1.code = prod_code
  RETURN rec.code as rec_code
}
```

The Recommendation Code

```
 6    RETURN p.code as prod_code
 7  }
 8  CALL {
 9  // For each of those purchases, return the 10 most similar
10  // by image embedding
11    USE fabric.products
12    WITH prod_code
13    MATCH (p1:Product)-[:SIMILAR]->(rec:Product)
14    WHERE p1.code = prod_code
15    RETURN rec.code as rec_code
16  }
17  CALL {
18  // For each of these recommendations, filter out the ones
19  // that do not come from a department or index previously
20  // purchases from by the customer
```

The Recommendation Code

```
18 // For each of these recommendations, filter out the ones
19 // that do not come from a department or index previously
20 // purchases from by the customer

21   USE fabric.neo4j
22   with rec_code
23   MATCH (c:Customer)
24   WHERE c.id = $customer_id
25   MATCH (ind:Index)<-[:HAS_INDEX]-(rec:Product)-[:FROM_DEPARTM
26   WHERE rec.code = rec_code
27   AND EXISTS ((c)-[:PURCHASED]->(:Product)-[:FROM_DEPARTMENT]-
28   AND EXISTS ((c)-[:PURCHASED]->(:Product)-[:HAS_INDEX]->(ind)
29   RETURN rec.code AS recommended_item
30 }
31
32 RETURN collect(recommended_item) as recommended_items
```

The Recommendation Code

Write all recommendations to a sparse matrix

```
for i, cust in enumerate(customers):
    customer_index = customer_to_idx[cust]
    recs = neo_recs.get_recommended_items_for_user(cust)
    rec_indices = [product_to_idx[rec] for rec in recs]
    rec_matrix[customer_index, rec_indices] = 1
```


MakeAG

The Final Part

The Final Part

1. For each customer, get the recommendations based off of image similarity and filtered with the graph model

The Final Part

1. For each customer, get the recommendations based off of image similarity and filtered with the graph model
2. Check they were part of our ALS model dataset - if they were, then rank all of the recs selected from part 1

The Final Part

1. For each customer, get the recommendations based off of image similarity and filtered with the graph model
2. Check they were part of our ALS model dataset - if they were, then rank all of the recs selected from part 1
3. If they were not, or if less than 12 items returned, randomly sample our recommendations from part 1

The Final Part

1. For each customer, get the recommendations based off of image similarity and filtered with the graph model
2. Check they were part of our ALS model dataset - if they were, then rank all of the recs selected from part 1
3. If they were not, or if less than 12 items returned, randomly sample our recommendations from part 1
4. Finally, if a customer has zero recommendations, use pure Cypher query

If images/ALS don't have the answer...

Cypher query for those with few items from previous query. Only used as backup query.

```
MATCH (c:Customer {id: $customer_id})-[:PURCHASED]->(p:Product)
[:PURCHASED]-(:Customer)-[:PURCHASED]->(rec:Product)
WHERE id(p) <> id(rec)
AND NOT EXISTS ((c)-[:PURCHASED]->(rec))
WITH c.id as customer_id, rec, COUNT(rec) as score
ORDER BY COUNT(rec) DESC LIMIT 12
RETURN collect(rec.code) as recommended_items
```

If images/ALS don't have the answer...

Cypher query for those with few items from previous query. Only used as backup query.

```
MATCH (c:Customer {id: $customer_id})-[:PURCHASED]->(p:Product)
[:PURCHASED]-(:Customer)-[:PURCHASED]->(rec:Product)
WHERE id(p) <> id(rec)
AND NOT EXISTS ((c)-[:PURCHASED]->(rec))
WITH c.id as customer_id, rec, COUNT(rec) as score
ORDER BY COUNT(rec) DESC LIMIT 12
RETURN collect(rec.code) as recommended_items
```

If images/ALS don't have the answer...

Cypher query for those with few items from previous query. Only used as backup query.

```
MATCH (c:Customer {id: $customer_id})-[:PURCHASED]->(p:Product)
[:PURCHASED]-(:Customer)-[:PURCHASED]->(rec:Product)
WHERE id(p) <> id(rec)
AND NOT EXISTS ((c)-[:PURCHASED]->(rec))
WITH c.id as customer_id, rec, COUNT(rec) as score
ORDER BY COUNT(rec) DESC LIMIT 12
RETURN collect(rec.code) as recommended_items
```

If images/ALS don't have the answer...

Cypher query for those with few items from previous query. Only used as backup query.

```
MATCH (c:Customer {id: $customer_id})-[:PURCHASED]->(p:Product)
[:PURCHASED]-(:Customer)-[:PURCHASED]->(rec:Product)
WHERE id(p) <> id(rec)
AND NOT EXISTS ((c)-[:PURCHASED]->(rec))
WITH c.id as customer_id, rec, COUNT(rec) as score
ORDER BY COUNT(rec) DESC LIMIT 12
RETURN collect(rec.code) as recommended_items
```

Eyeballing The Recommendations Part 2

What would I do differently?

- Change the graph model/fabric setup
- Optimize ALS model
- Optimize the inference process
- Utilise more of the customer data