GRec: A Graph-Based Recommendation System for Recipes

DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA UNIVERSITÀ DI ROMA

Master in Artificial Intelligence and Robotics

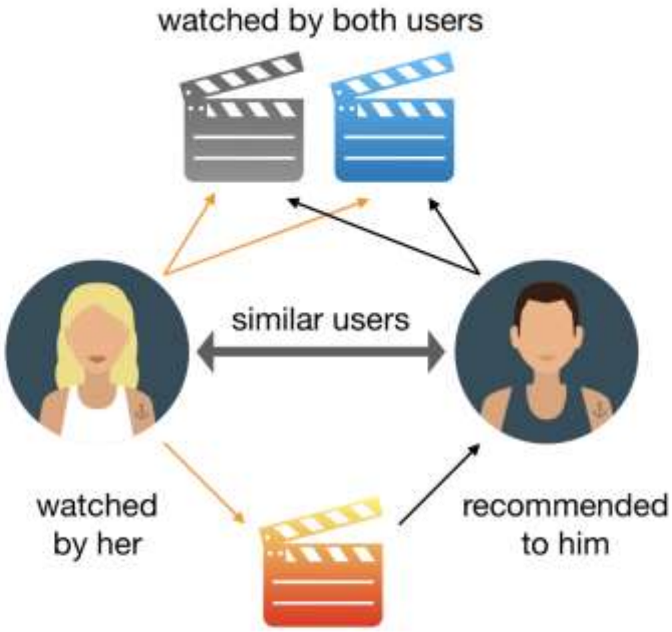
Course	Data Management
Professor	Maurizio Lenzerini
Tutor	Roberto Maria Delfino
Student	Luigi Gallo (1895146)

Outline

Introduction

The use of the Internet as a **business** tool has necessitated the development of technologies to **suggest** products of interest to **consumers**.

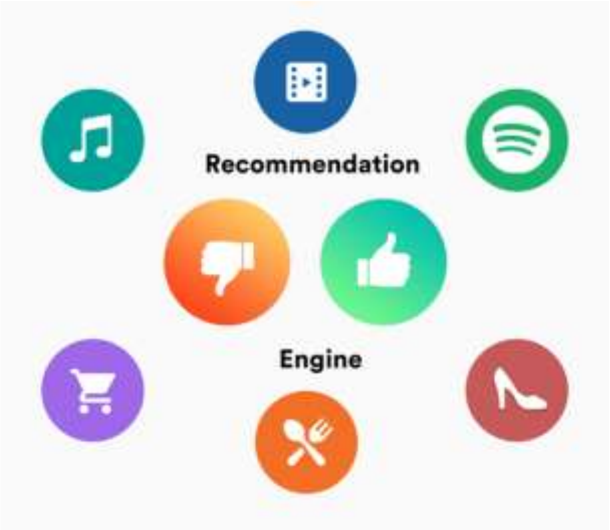
By analyzing interactions with such **products**, current recommendation systems are able to provide **accurate** suggestions.



Introduction

What are Recommender Systems

Recommender Systems are a type of information filtering system that seek to generate *meaningful* recommendations to ***users*** for ***items*** they may be interested in.



The most used filtering techniques are **collaborative** and **content-based** filtering.

Introduction

Types of Recommender Systems

Collaborative Filtering refers to the use of **ratings** from multiple **users** in a collaborative way to predict missing ratings.

Advantages: recommend complex items without **understanding** the item.

2 **types**:

1. *User*-based
2. *Item*-based

Introduction

Types of Recommender Systems

In **content-based** recommender systems, the *descriptive* attributes of **items** are used to make recommendations.

The critical premise of content-based filtering is that if you like an item, you will also like a **similar item**.

Project Overview

About the project

The **goal** of this project is to, starting with the choice of a **dataset**, implement a **recommendation system** based on **graph** databases.

Data analysis will be needed to **understand** how the **data** are distributed and what information to leverage to suggest relevant recipes to users.

This will help us define the **schema** of the **database** keeping in mind the type of **queries** that will be executed.

Dataset Overview

About the dataset

The **dataset** is '**Food.com Recipes and Interactions**'.

- **180K+** recipes
- **700K+** reviews
- **18 years** of user interactions and uploads

In addition to using classic recommendation methods to suggest recipes, it is also possible to **filter** the results according to user-defined preferences.

Dataset Overview

RAW_recipes.csv

name	id	minutes	contributor_id	submitted	tags
Recipe name	**Recipe ID**	**Minutes to prepare recipe**	**User ID who submitted this recipe**	**Date recipe was submitted**	**Food.com tags for recipe**
arriba baked...	137739	55	47892	2005-09-16	['60-minutes-or-less', 'occas...

nutrition	n_steps	steps	description	ingredients	n_ingredients
Nutrition information	**Number of steps in recipe**	**Text for recipe steps, in order**	**User-provided description**	**List of ingredient names**	**Number of ingredients**
[51.5, 0.0, 13.0, 0.0, 2.0, 0.0, 4.0]	11	['make a choice and proceed with recipe', ...	autumn is my favorite time of year to cook!...	['winter squash', ...	7

Dataset Overview

RAW_interactions.csv

user_id	recipe_id	date	rating	review
User ID	**Recipe ID**	**Date of interaction**	**Rating given**	**Review text**
38094	40893	2003-02-17	4	Great with a salad ...

Leverage user preferences to **filter** all possible recipes proposed by collaborative-filtering.

Data Analysis

Analyze data to understand the dataset

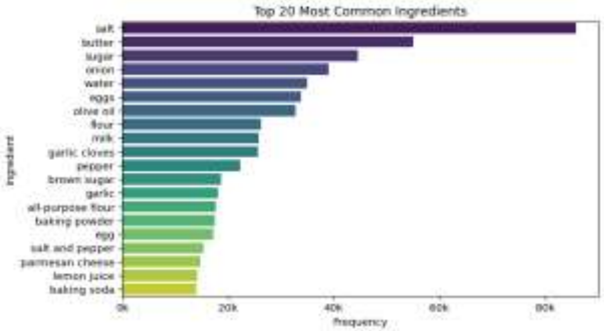
Before diving into the database creation, it's *crucial* to **understand** the dataset thoroughly and **identify** any **patterns** or characteristics that might inform our *recommendation* ***strategy***.

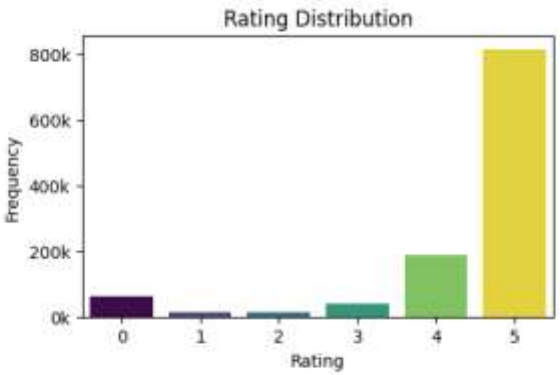
This understanding can **help** in **refining** the **criteria** for selecting recommendations.

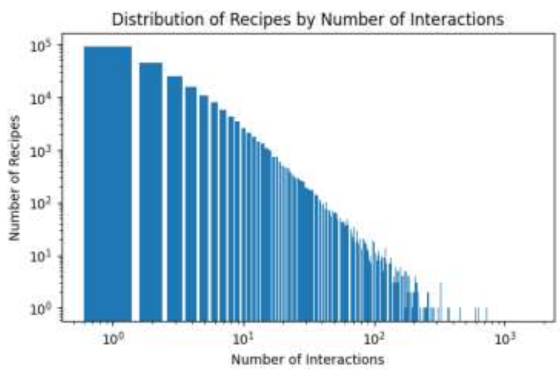
Data Analysis

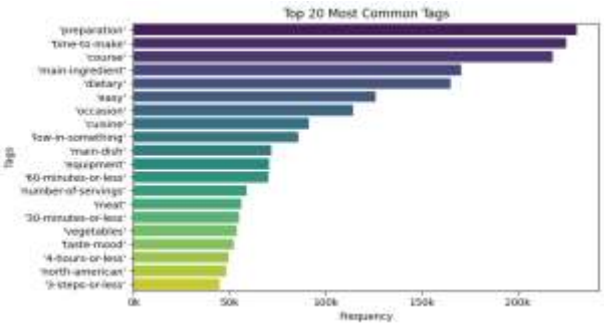
Rating distribution

Users generally **rate recipes they like** and *might not bother rating others*.
Content-based filtering might be the best choice.
Common tags and ingredients may be skipped.





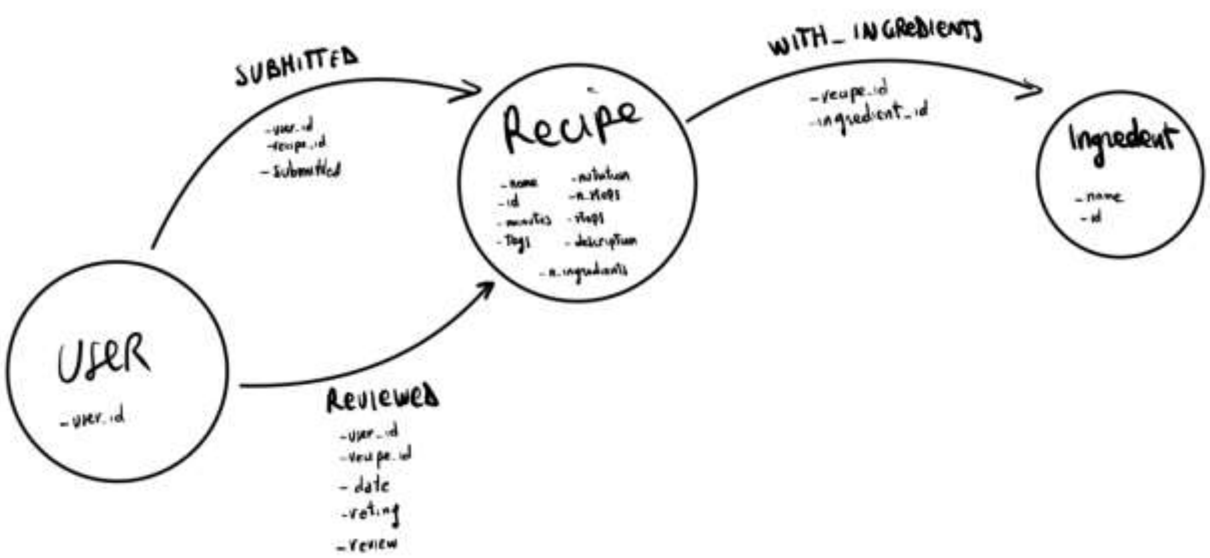




Data Preprocessing

How to prepare data

Before importing data into **neo4j**, it is necessary to handle missing (**null**) values and **format** the data in tables designed to model each *Node* or *Relationship*.



Data Preprocessing

Data files

recipes.csv

name
id
minutes
tags
nutrition
n_steps
steps
description
n_ingredients

users.csv

user_id
recipe_id
date
rating
review

reviewed.csv

recipe_id
user_id
submitted

submitted.csv

id

ingredients.csv

id
name

with_ingredients.csv

recipe_id
ingredient_id

Neo4j

Why a graph database?

Graph databases are **ideal** for building a recommender system for several reasons:

1. **Real-Time** Recommendations
2. **Fast** to Develop, Maintain and Expand
3. **Scalability**
4. Highly **Performant**
5. Graph **Algorithms**

Neo4j

GDS (Graph Data Science)

The Neo4j Graph Data Science (**GDS**) library is a comprehensive **plugin** for performing advanced graph analytics and machine learning on Neo4j databases.

It's used to execute GDS algorithms and **queries** against your Neo4j **database**, integrating graph-based insights and analytics directly into **Python** applications and data pipelines.

Neo4j

APOC (Awesome Procedures On Cypher)

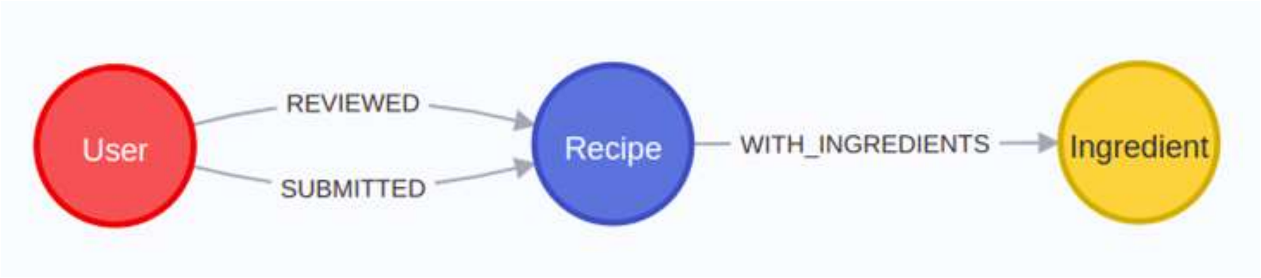
The Neo4j **APOC** (Awesome Procedures On Cypher) library is a collection of pre-built *procedures* and *functions* that **extend** the **capabilities** of Neo4j's Cypher query language.

- **`apoc.convert.fromJsonList(str)`**: convert a JSON string into a Cypher list.
- **`CALL apoc.meta.stats()`**: retrieve statistics about the nodes, relationships, and properties within a Neo4j database.

Building a Recommendation System

Resulting schema

After properly processing the data, setting uniqueness constraints (**keys**), and creating nodes and relationships, the resulting **schema** is as follows:



Queries

Get user's interactions

```
MATCH (u:User {id: $userID})-[i:SUBMITTED|REVIEWED]->(r:Recipe)
RETURN
  r.id AS recipeID,
  r.name AS name,
  r.nutrition AS nutrition,
  r.n_ingredients AS n_ingredients,
  COALESCE(i.submitted, i.date) AS interactionDate,
  type(i) AS interactionType
ORDER BY interactionType, interactionDate DESC
```

Queries

Get user's interactions

The query returns all the **interactions** (submissions and reviews) of a given user, specified by the **`$userID`**.

	recipeID	name	nutrition	n_ingredients	interactionDate	interactionType
0	297700	beer battered string green beans with remoulade dip	[450.6, 31.0, 42.0, 53.0, 15.0, 15.0, 19.0]	19	2009-07-25	REVIEWED
1	382430	kfc s 11 herbs and spices	[321.7, 15.0, 31.0, 785.0, 30.0, 9.0, 22.0]	11	2009-07-22	REVIEWED
2	174747	big ol mess smoked sausage in spicy sweet sauce	[426.9, 50.0, 44.0, 53.0, 31.0, 53.0, 5.0]	6	2009-07-18	REVIEWED
3	379531	trs rapide french summer tarragon chicken	[283.0, 26.0, 0.0, 5.0, 60.0, 31.0, 0.0]	7	2009-07-03	REVIEWED
4	371994	lizzie s dipping sauce	[68.8, 7.0, 9.0, 9.0, 0.0, 3.0, 2.0]	8	2009-06-17	REVIEWED

Queries

Get user’s favorite ingredients

```
MATCH (u:User {id: $userID})-[rel:REVIEWED|SUBMITTED]->(r:Recipe)
  -[:WITH_INGREDIENTS]->(i:Ingredient)
WHERE (rel.rating >= 4 OR TYPE(rel) = 'SUBMITTED')
  AND NOT i.name IN $excluded_ingr
WITH i, COUNT(r) AS favCount, COLLECT(r.id) AS favRecipes
ORDER BY favCount DESC
RETURN i.name AS favoriteIngredient, favCount, favRecipes
```

Queries

Get user’s favorite ingredients

The query returns the **favorite ingredients** of an user excluding **`$excluded_ingr`**. User favorite ingredients are extracted from the recipes the user has published and from the recipe the user evaluated with 4 or more rating.

	favoriteIngredient	favCount
0	tomatoes	28
1	garlic powder	23
2	garlic clove	20
3	mayonnaise	18
4	bacon	17
...	...	...
15	vegetable oil	13
16	red pepper flakes	11
17	parsley	11
18	bread	11
19	black pepper	11

Queries

Get user’s top tags

```
MATCH (u:User {id:$user_id})
  -[rel:SUBMITTED|REVIEWED]->(r:Recipe)
WHERE (rel.rating >= 4 OR TYPE(rel) = 'SUBMITTED')
WITH r, apoc.convert.fromJsonList(r.tags) AS tagList
UNWIND tagList AS tag
WITH tag
WHERE NOT tag IN $excluded_tags
RETURN tag, COUNT(*) AS tagCount
ORDER BY tagCount DESC
```

Queries

Get user’s top tags

The query returns the tags of an user excluding **`$excluded_tags`**.

	tag	tagCount
0	comfort-food	68
1	vegetarian	54
2	low-carb	54
3	beginner-cook	53
4	inexpensive	49
...	...	...
24	seafood	29
25	appetizers	24
26	poultry	24
27	tomatoes	23
28	summer	22

Queries

Get user recipe's nutritional values

```
MATCH (u:User {id:$userID})-[rel:SUBMITTED|REVIEWED]->(r:Recipe)
WHERE (rel.rating >= 4 OR TYPE(rel) = 'SUBMITTED')
WITH r, apoc.convert.fromJsonList(r.nutrition) AS nutritionList
RETURN  r.id AS recipeID,
     nutritionList[0] AS calories,
     nutritionList[1] AS totalFat,
     nutritionList[2] AS sugar,
     nutritionList[3] AS sodium,
     nutritionList[4] AS protein,
     nutritionList[5] AS saturatedFat,
     nutritionList[6] AS carbs
```

Queries

Get user recipe's nutritional values

The query returns the **nutritional values** of the recipes that the user has interacted with.

	recipeID	calories	totalFat	sugar	sodium	protein	saturatedFat	carbs
0	218117	71.8	0.0	57.0	30.0	3.0	0.0	5.0
1	226062	586.3	51.0	18.0	35.0	57.0	95.0	14.0
2	225973	154.7	13.0	37.0	8.0	2.0	22.0	6.0
3	268422	769.7	53.0	342.0	4.0	12.0	83.0	38.0
4	225975	509.1	69.0	101.0	22.0	1.0	160.0	8.0

Queries

Get recipes with specific **tags**, **nutr. value** and **ingredients**

```
MATCH (r:Recipe)-[:WITH_INGREDIENTS]->(i:Ingredient)
WHERE NOT r.id IN $interactedRecipes
WITH
  r, COLLECT(i.name) AS recipeIngredients,
  apoc.convert.fromJsonList(r.nutrition) AS nutritionList,
  apoc.convert.fromJsonList(r.tags) AS tagList
WHERE
  ANY(ingredient IN recipeIngredients WHERE ingredient IN $favIngreds)
  AND ANY(tag IN tagList WHERE tag IN $topTags)
  AND nutritionList[0] > $min_0 AND nutritionList[0] < $max_0      // CALORIES
  AND nutritionList[1] > $min_1 AND nutritionList[1] < $max_1      // TOTAL FAT (PDV)
  AND nutritionList[2] > $min_2 AND nutritionList[2] < $max_2      // SUGAR (PVD)
  AND nutritionList[3] > $min_3 AND nutritionList[3] < $max_3      // SODIUM (PDV)
  AND nutritionList[4] > $min_4 AND nutritionList[4] < $max_4      // PROTEIN
  AND nutritionList[5] > $min_5 AND nutritionList[5] < $max_5      // SATURATED FAT (PDV)
  AND nutritionList[6] > $min_6 AND nutritionList[6] < $max_6      // CARBS
RETURN
  r.id AS recipeID, r.name AS recipeName,
  SIZE([ingredient IN recipeIngredients WHERE ingredient IN $favIngreds]) AS matchingIngreds,
  SIZE([tag IN tagList WHERE tag IN $topTags]) AS matchingTags,
  (toFloat(SIZE([ingredient IN recipeIngredients WHERE ingredient IN $favIngreds]))
  / SIZE(recipeIngredients) * log(1 + SIZE(recipeIngredients))) AS ingrRelScore,
  (toFloat(SIZE([tag IN tagList WHERE tag IN $topTags])) / SIZE(tagList)
  * log(1 + SIZE(tagList))) AS tagRelScore
ORDER BY (ingrRelScore + tagRelScore) DESC
```

Queries

Get recipes with specific **tags**, **nutr. value** and **ingredients**

The query returns recipies that have the same **tags** as **`$top_tags`** and **ingredients** as **`$favorite_ingredients`**, that respect certain **nutritional values** criteria, and excluding from those **`$interacted_recipes`**.

	recipeID	recipeName	matchingIngreds	matchingTags	ingrRelScore	tagRelScore
0	451220	kohlrabi and carrot slaw	6	8	1.438737	1.360175
1	40925	celery nut salad	3	13	1.075056	1.604395
2	503674	german style tomato salad	4	8	1.098612	1.308640
3	316505	stir fry parmesan yellow squash	2	9	0.648637	1.740889
4	188160	mexican night salad	4	9	0.854983	1.472219

About relevance

Top ingredients and top tags

To obtain top ingredients and top tags that were actually relevant, considering either a **fixed** count **threshold** or a **percentage** of the result is **not** a **good** approach. If **many** ingredients or tags have a count of **1**, taking for example the top 50% ingredients (tags) sorted by count could include many of these with low counts. Establishing a fixed count threshold, on the other hand, might not be an appropriate choice anyway, since users with **more reviews** generally have **higher** ingredient (tag) **counts**.

For these reasons, the choices made in determining the top ingredients (tags) fall back to **turning counts into sets** (of counts), and taking a **50 percentile** of this as the **threshold**. This provides a dynamic threshold and a better result.

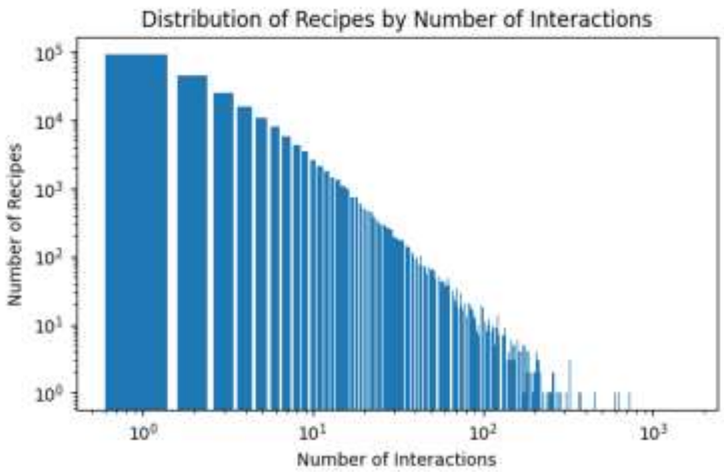
Performance

How to assess performance

To measure performance, the interactions data were **divided** into **two** parts: one to populate the database and the other to test it.

After creating the graph database, the previous queries are used to produce a list of user recommendations, comparing how many of them appear in the **test data**.

Unfortunately, due to the very **complex** nature of the **dataset** which is very **unbalanced**, the performance obtained is **poor**.



Thank you for your attention