Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Data Science

Development of a Healthy Food Recommendation System over Semantic-Driven Knowledge Graph

Relatore: Prof. Paolo Napoletano
Correlatore: Prof. Gianluigi Ciocca

Tesi di Laurea Magistrale di:
Giorgia Rigamonti
Matricola 844619

Anno Accademico 2022 - 2023

Abstract

Food is fundamental for human beings, as it is indispensable for health and, above all, for life itself. In recent years, the recording and sharing of recipes in online repositories has led to a rapid growth of food computing: there is a large-scale availability of online recipe collections that offer users the opportunity to access a wide variety of recipes. However, this can be challenging and can make it difficult for users to discover recipes relevant to their request. In order to meet the customized needs of users, Recommendation Systems have been developed.
Compared to other Recommendation Systems applied to general domains such as films, music or products, food recommendation is highly relevant to human health and, therefore, plays a critical role in human food choices. Food recommendation involves more complex and multifaceted information: in addition to satisfying user preferences, it is therefore equally important to include health scores, which is why Food Recommendation Systems have recently received increasing attention due to their importance for a healthy lifestyle.
Added to this is the introduction of knowledge graphs (KG) into Recommendation Systems in recent years as collateral information. It has attracted significant interest as it can enable a Food Recommendation System to understand latent reciprocal relationships better and make recommendation results more explainable.
Nevertheless, its use in food recommendation has not been extensively studied, and to fill this gap, this thesis work aims to describe the implementation of a Food Recommendation System based on a knowledge graph that can provide users with recommendations that meet their own customized requirements, while considering the healthiness of recipes and the user's unique health conditions, such as food allergies or specific food diets.
To achieve this, a Food Knowledge Graph is reconstructed and, starting from its initial structure, a mechanism is defined for inserting new characteristics aimed at further extending the created FoodKG with new labels that consider, in the ingredients that make up a recipe, their healthiness, the possible presence of allergens or their unsuitability for particular food diets.
From this, it is possible to implement a content-based Recommendation System whereby the recommendation problem is seen as a binary classification problem in which the intention is to predict which recipes the user might or might not like. In this case, in order to cope with the scarcity of user interaction with the recipes, semi-supervised learning, and self-training techniques are applied, which resulted in the recommendation of the six recipes most suitable for the user.

Finally, a Web App is developed and implemented through which users can interact with the recipe Recommendation System created.

Acknowledgements

First of all, I want to thank all the people who made this thesis possible.
I would like to thank my professors Paolo Napoletano and Gianluigi Ciocca, who have always been available and have followed and guided me during this period.
Special thanks go to my parents and my sisters Micol, Cristina and Alessia, who have always supported and sustained me.
Thanks to my friends and classmates who I met during these years at university, in particular, Valentina, my partner in these two years of lessons, projects and adventures.
I also have to thank Gabriele for having always been close to me, both in the most carefree moments but especially in the most difficult ones, and for having always encouraged and stimulated me to reach my goals.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The problem of the proliferation of information and content that can be found, especially within Web platforms, has led in recent years to an ever-increasing demand for systems that allow rapid access to them, cutting down search times.
Indeed, the evolution of information systems and the widespread diffusion of the Web have made such an impressive amount of information available to users that it is a challenge regarding cognitive overload. In other words, a user's ability to make a decision may be compromised by the presence of too much information.
On the other hand, an overload of resources often leads to long searches to find what one likes best. Glaring examples can be found in everyday life: great demand and the consequent production of television series, disproportionate increase of online sources for the most varied themes and topics, disproportionate increase in the availability of video and music content, escalation of social networks and the information they contain, proliferation of articles available within e-commerce platforms, ...
Taking everything into account becomes, in fact, impossible.
In order to solve this problem, Recommendation Systems are born, namely software, often integrated into applications, which, by exploiting prior knowledge of the user's tastes or explicit feedback provided by the user, create customized recommendations attempting to facilitate their choices, thus helping them to find only the content that best meets their needs.
The recommendation is applied in multiple decision-making processes, such as what objects to buy, what music to listen to or what news to read, ...

In effect, online Recommendation Systems have proven to be helpful in several situations, enabling users to overcome the problem of information overload, assisting them in

the decision-making process and simultaneously changing their behaviour. However, one domain that has historically received little attention, especially when compared to other domains, is Food Recommendation. This is surprising considering the importance of food for human sustenance.
Moreover, food plays a highly crucial role in the health of individuals: making healthier dietary decisions can be crucial for daily well-being and is indispensable for preventing and managing chronic diseases such as diabetes, hypertension, cancer, mental illness, asthma,...
In recent years, people have become increasingly aware of the importance of their health. The significant improvements in living conditions worldwide have brought about this increased awareness.
With increasing access to food resources and nutritional knowledge, people are challenged to make more informed and conscious decisions about their health and well-being. Dietary advice has the potential to influence people's eating habits positively.
In this regard, Food Recommendation Systems have been developed to provide personalized and automated recommendations. By harnessing the power of Machine Learning algorithms, Food Recommendation Systems are able to analyze user data and make informed recommendations. Food Recommendation Systems not only provide relevant recommendations that users may want to eat but also help users to consume a healthier diet.
However, several reasons make food recommendations difficult, such as predicting what people would like to eat because it is often complicated, multifaceted, culturally determined and context-dependent. For this reason, health-conscious Food Recommendation Systems are often considered an essential part of the solution to encourage healthy nutritional choices.
Moreover, there are many reasons why food recommendation is a complex task that presents unique challenges. Firstly, since food recommendations are personalized, it is necessary to ensure they are diverse and relevant to the user. There is often more than one optimal recommendation, depending on the circumstances. Secondly, since food recommendation is a dynamic task, it is necessary to adapt quickly and accurately to the user's preferences, which must consider the user's dynamic behaviour when considering appropriate recommendations. Last but not least, it is necessary to accurately capture and represent user preferences, which can be challenging because preferences are often subjective.
In recent years, introducing Knowledge Graphs (KG) into the Recommendation System has attracted great interest because a KG can provide additional information and suggest potential relationships between items. In particular, recipe items and their attributes (i.e. ingredients and categories) are mapped in the KG can enable a Food Recommendation System to better understand the latent mutual relationships between them and make the recommendations' results more explainable. However, recipe recommendation based on

KG has yet to be extensively studied.
To this end, this thesis aims to develop a Personalized Knowledge Graph-based Recommendation System that can recommend recipes that meet the user's preferences but also take into account the user's health based on the Food Standards Agency's nutrition standard. Such a Recommendation System considers the user's explicit needs and crucial health factors, such as any food allergies and lifestyle preferences, such as the desire to eat only vegan or vegetarian food.

Structure of the Thesis

In Chapter 2, an initial overview of Recommendation Systems will be given, including a brief history of Recommendation Systems, a description of the different types of existing Recommendation Systems, the main problems associated with them and the associated evaluation methods. In Chapter 3, the first steps towards the implementation of a Food Recommendation System will be described: initially, the use of knowledge graphs in Recommendation Systems will be presented with a focus on Food Recommendation Systems; subsequently, the procedure for constructing a large knowledge graph and its extension process will be shown. In Chapter 4, the implementation of a Food Recommendation system based on knowledge graphs will be discussed to consider the user's preferences and the wholesomeness of recipes by incorporating the presence of any food allergies and the adoption of a particular food diet. Chapter 5 will present the construction of a Web App that will allow the Recommendation System to interface with the user. Finally, Chapter 6 will provide some considerations on the results and possible future developments.

Chapter 2

The Background of Recommendation Systems

This chapter will provide a detailed overview of Recommendation Systems.
It will start with the definition of the recommendation problem and then describe the main types of Recommendation Systems currently used, the associated problems, and the evaluation measures adopted to calculate their performance.

2.1 A Recommendation Systems Overview

Before the Internet and Search Engines spread, finding helpful information for a specific topic or context took much work, time-consuming, and tiring to search. It was often expensive as buying books or traveling to consult libraries was shared.
Today, the world has changed a lot, and thanks to the many search engines, thousands upon thousands of data and information are immediately available to everyone. On the other hand, there is a growing need to learn how to navigate this sea of information, not to be frightened or demoralized by so many results, and to learn to search. In fact, it is becoming increasingly complicated to find the correct information at the right time, to be able to buy the right dress for an event in the shortest amount of time, to be able to choose from the myriad of films available, the one that best suits our mood,...
This critical issue gave rise to the first Recommendation Systems, software tools designed to assist the user in selecting the resources most aligned with their tastes and needs and most relevant to the search and context.

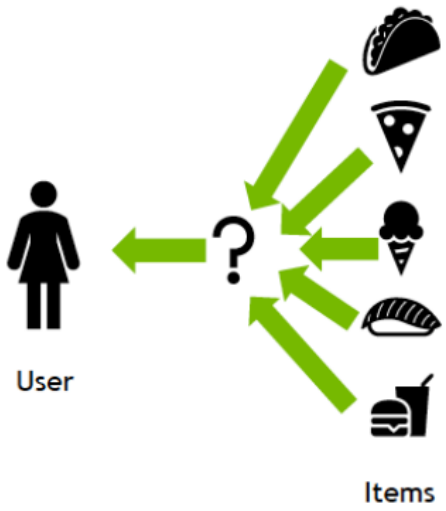

Figure 2.1: The Recommendation System

More precisely, Recommendation Systems developed with the proliferation of e-commerce, namely online shops with more extensive catalogs and more alternatives. Given their vastness, the main problem associated with them was their size, which entailed more incredible difficulty in the decision-making process, and thus the risk of making a wrong choice: in fact, while having more options to choose from is undoubtedly a good thing, having too many can be counterproductive.
Nowadays, one can observe the presence of these systems in a variety of fields: they help the user choose which film to see, which music to listen to, which product is suitable for their needs, which news may interest them, which people to add among friends in social networks,...
Providing impersonal suggestions on popular items is a relatively simple problem. Still, it is not necessarily the case that all users like the same items: it is precisely in the ability to provide personalized suggestions, responsive to the tastes of the individual user, that lies the complexity of an efficient and effective Recommendation System and to be so, there are several factors that it must take into account, including past purchases, search history, demographic information, clicks,...
In this way, the recommendations of these systems can have strong persuasive power and succeed in influencing the user's choices and, in some cases, changing the product life cycle, helping the user to discover products and services that they would not otherwise have found on their own, while, at the same time, increasing their level of engagement and

satisfaction.
In this regard, it is worth mentioning the case of a book titled "Touching the Void," which was not very successful at the time of publication. However, years later, with the publication of another book entitled "Into Thin Air," which dealt with the same subject matter as the first, Amazon's recommendation algorithm noticed that there were some individuals who had purchased both books and began recommending "Touching the Void" to people who had bought or had shown interest in the book "Into Thin Air" this fostered the success of "Touching The Void," a book that without Amazon's algorithm could almost certainly not have found any potential buyers.

2.2 The Recommendation Problem

A Recommendation System is a technology used in an environment in which objects (products, movies, events, items) are recommended to customers (clients, visitors, users, readers) or the other way around. Typically, many objects and users are present in the environment of a Recommendation System, making the problem complex and expensive.
The Figure 2.2 shows this problem can be solved in three main steps: information gathering, learning and inference (prediction or recommendation).

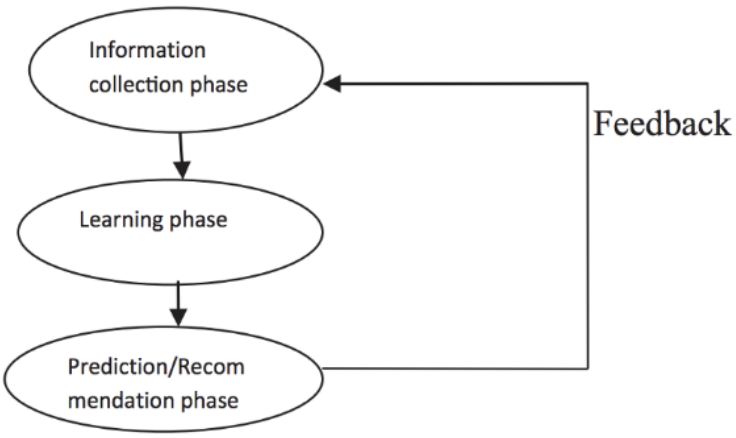

Figure 2.2: The Recommendation System Steps

In the first phase, the Information Retrieval phase, the system have to learn about the user by observing and recording how the user interacts with the system and the elements.

This is why relevant user information is collected so that a profile of the user can be generated to serve as a model for prediction tasks, including user attributes, behaviors, or the content of the resources the user accesses. The system needs to know as much as possible from the user to provide him with useful and reliable recommendations.
For this purpose, Recommendation Systems rely on different input types called feedback, which can be stored so that it can be used later to generate new recommendations.
There are two different techniques for recording user feedback:

- **Explicit Feedback:** occurs when a system requires the user to evaluate items explicitly. There are three different types of explicit feedback:
  - *Binary:* when items are rated as relevant or irrelevant according to a binary scale;
  - *Ratings:* when a numerical scale is used to rank items;
  - *Text Comments:* when text comments are used to classify items. They allow users to write text comments on items, thus helping new users in decision-making. They contain much information but are difficult for an automated system to process. Advanced sentiment analysis software is needed to analyze whether the comment is positive or negative toward the article and to what extent.

  These methods are easy to apply, but explicit feedback generally involves more cognitive load for the user. It may not even be able to capture their feelings toward an article.

- **Implicit Feedback:** occurs when a system does not require active involvement on the part of the user. In this case, to make recommendations, how the user interacts with the system is analyzed by scoring different user actions on an item. This type of feedback is more pronounced because users are generally unaware of the process taking place, but over time, the system learns more and more about the user's true preferences.

Once the feedback is obtained and stored in the preferred mode, the system applies a learning algorithm to filter and exploit user characteristics from the input collected in the Information-Gathering phase. With this pattern discovered during the Learning phase, the system can finally interpret the user's preferences. The task can be to make a prediction or recommendation: the system can predict feedback from the user on a particular item and then use this information to decide whether to suggest the item in question to the user

or try to select items at the top of the list of those the user might be interested in.

A more mathematical definition of the Recommendation Problem was given by Sarwaretal et al. in 2001 [36]. The Prediction Problem can be formalized as follows: let

$$U = u_1, u_2, ..., u_m$$

be the set of all users, and let

$$I = i_1, i_2, ..., i_n$$

be the set of all possible items that can be recommended. Each user u_i has a list of items I_{u_i}. This list represents the items that the user has expressed their interests. Note that $I_{u_i} \subseteq I$, and it is possible that $I_{u_i} = \varnothing$, such as in the case of new subscription to a service. Then, the function, $P_{u_i} i_j$ is the predicted likeliness of item i_j for the active user u_a, such as $i_j \notin I_{u_i}$. The Recommendation Problem can be stated as obtaining a list of N items, $I_r \subset I$, that the user will like the most (i.e the ones with higher $P_{u_a} i_j$ value).
The recommended list should not contain items from the user's interests, therefore $I_r \cap I_{u_i} = \varnothing$. Both the set I of possible items and the set of users U can be very large. In most recommendation systems, the prediction function $P_{u_a} i_j$ is usually represented by a rating, which is given by the user either explicitly or implicitly through some measures, for example, by tracking if a movie is skipped in the first minutes of play. They are represented as triplets $< h_u,\ i,\ r_i >$ where r is the rating value assigned by the user u to a particular item i. The value is usually a real number (e.g., from 0 to 1), a value in a discrete range (e.g., from 1 to 5), or a binary variable (e.g., like/dislike).

2.3 Types of Recommendation Systems

When implementing a Recommendation System, there are several options to choose from regarding the exact way in which recommendations are made.
As can be seen in the Figure 2.3, Recommendation Systems can be divided mainly into two macro-categories: *Collaborative filtering methods*, which calculate recommendations by taking into account the ratings on various elements left by similar users and *Content-based methods*, which, instead, rely on the content, on the characteristics of an element in relation to those of the user, thus suggesting elements that are similar in content to others previously seen by the user. Furthermore, these two approaches can be combined to give rise to *Hybrid solutions* that exploit the advantages of both.
The main types of Recommendation Systems will be described in more detail below.

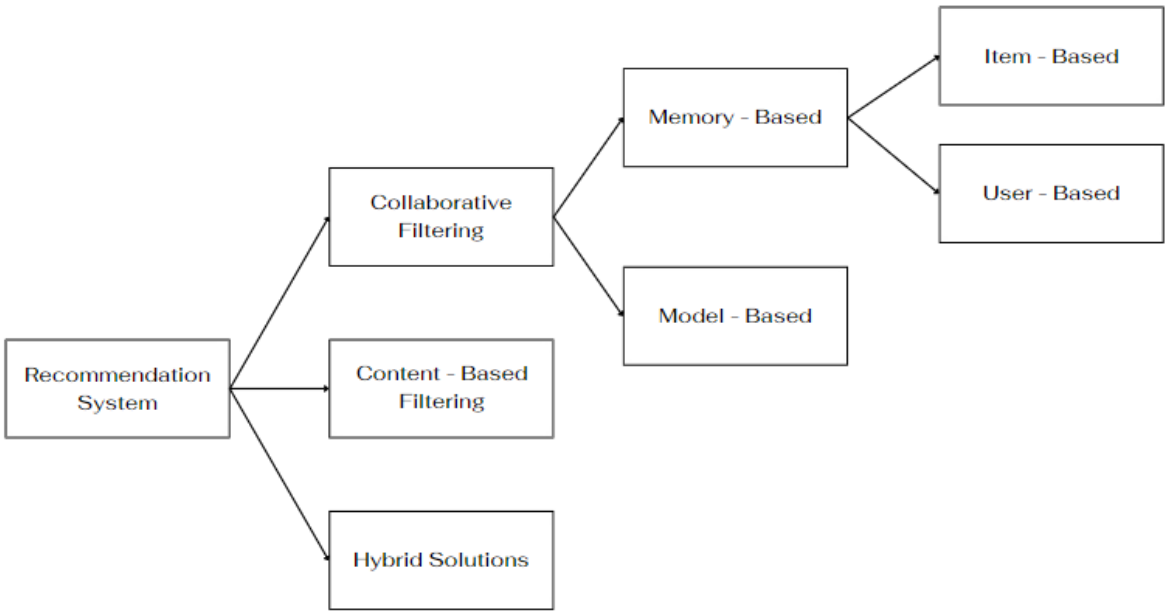

Figure 2.3: Outline of the types of Recommendation Systems

2.3.1 Content - Based Filtering

Content-based filtering (CBF), as shown in the Figure 2.4, uses the attributes specified by the user or item as the basis for comparison, which is used to determine relevance. By exploiting behavioral history, CBF can identify and recommend items likely to interest users. The result of the recommendation is a correlation that reflects the user's degree of preference for the item. The idea behind content-based filtering is that if the user likes an item, they are likely also to enjoy other items with the same characteristics. The three components of CBF can be summarized as follows:

- *Content Analyzer:* when data is presented in an unstructured format, a processing step is used to obtain structured and meaningful information. The analyzer converts the input data (e.g., documents, web pages, news, product descriptions,...).
- *Profile Learner:* by capturing descriptions or attributes of objects the user has interacted with, the *profile learner* develops a profile incorporating the user's interests and preferences.
- *Filtering Component:* in which recommendations are made by finding similar items

that users have liked in the past.

To summarize, the content-based approach consists of evaluating the attributes of items the target user has liked and suggesting their new items with these attributes.
On the other hand, a content-based approach is largely based on available information, with two potential disadvantages: limited content analysis, which refers to restricted content and the amount of information available. In these cases, the accounting will always be more accurate than the available content, which may be superficial or inaccurate; and over-specialization, which occurs when the content-based approach focuses too much on a limited area and, as a result, fails to capture the full scope of the data.

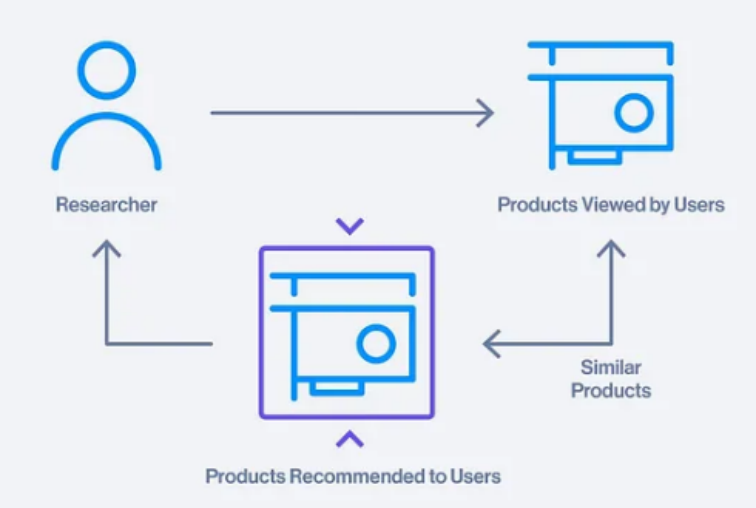

Figure 2.4: Scheme of Content - Based Filtering Recommender Systems

Companies taking advantage of such an approach can be found in e-commerce sites, but also in portals for choosing rental cars, online insurance and other cases where it is essential to take into account not only the historical characteristics of a user but, above all their changing behaviour, needs and tastes that change over time.
An example of applying a content-based approach is the "You might also like:" section of Zalando, where the system offers suggestions of objects similar in content (shape, colour, fantasy, ...) to the current search.

2.3.2 Collaborative Filtering

Unlike CBF, collaborative filtering (CF) techniques exploit similarity among users and assume that groups of similar users also have similar tastes.
The selection of suggestions is based primarily on the type of items previously seen by the user and the ratings (through votes, feedback, likes, ...) that the user has attributed to each item. As shown in the Figure 2.5, assuming that similar users attribute a similar rating to the same element, the recommendation algorithm provides the user with suggestions for the features to which the similar users have associated the highest ratings.
The idea behind this approach is that historical user data are sufficient to make a prediction, that nothing else needs to be known in order to identify the suggestions to be proposed to the user.
This recommendation process consists of three steps:

1. Users are grouped according to their user profiles;
2. A ranking of the most popular items within the cluster is drawn up;
3. The top item that the user has yet to see before is recommended to the user.

The main advantage of CF is that it can learn about the articles to make recommendations. Recommendations, in fact, are based solely on user similarities, meaning that complex items can be recommended. However, the lack of information about the content can be a disadvantage.

Classic examples of the application of this type of Recommendation System are Facebook and Linkedin, which, through historical analysis of user interactions and evaluations with content and people on the network, are both able to propose new content to each user every day that, even if it does not concern people, companies or groups to which the user is directly connected, may be of interest because similar users like it.
Another example of collaborative filtering is the "Who bought this item also bought it" section proposed by Amazon: in this case, user suggestions are calculated based on the reactions, feedback and purchases of similar customers on the platform.

This class of methods can be further divided into two subcategories based on the technique used to identify similarities between users and/or items:

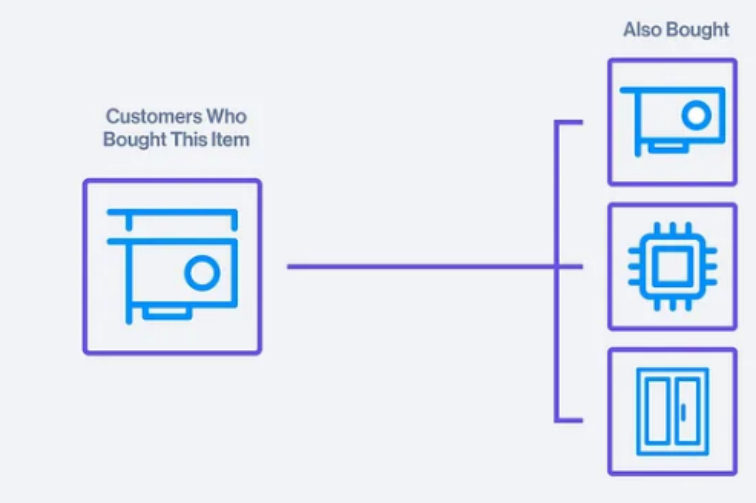

Figure 2.5: Scheme of Collaborative Filtering Recommender Systems

- **Memory - Based approach:** predicts item ratings based on all ratings provided by different users for an item. This method can in turn be subdivided into:
  - *Item - Based method:* represents items based on the interactions users have had with them. Two items are considered similar if most users who interacted with both did so in the same way. To make a new recommendation to a user, these methods search for items similar to those with which the user has positively interacted.
  - *User - Based method:* represents users by considering their interactions with items and based on this, they assess the similarity between one user and another. In general, two users are considered similar if they have interacted with many items in the same way. To make a new recommendation to a user, an attempt is made to identify those with the interaction profiles most similar to his to suggest the most popular item in his neighborhood.
- **Model - Based approach:** predicts user ratings for all items from a specific collection of user-rated articles.

The main advantage of collaborative filtering methods is that they do not require the extraction of user or item information and can, therefore, be used in various contexts. In addition, the more users interact with items, the more information will be available and the more accurate the new recommendations will be.

Their disadvantage emerges when you have new users or new items because there is no past information about their interactions; this situation is called the cold start problem. In this case, several techniques are used to determine what the new recommendations should be: recommending randomly chosen items to new users or new items to randomly chosen users, recommending popular items to new users or new items to the most active users, recommending a set of several items to new users or a new item to a set of several users, or, avoiding a collaborative filtering approach at this stage.

2.3.3 Hybrid Solutions

The *Hybrid Method*, as shown in the Figure 2.6, is based on the idea that the combination of these algorithms offers more accurate and precise recommendations than a single algorithm, since the shortcomings of one algorithm can be addressed by another algorithm.
In a combined model, using multiple recommendation approaches can help mitigate the shortcomings of each strategy. To address model weaknesses, hybrid models attempt to integrate more than one model commonly used in practice, even if this results in increased complexity due to the integration of separate techniques and algorithms.
They can be combined in various ways, including using content-based filters in a collaborative approach, implementing algorithms separately and combining the results, implementing collaborative filters in a content-based approach and creating a unified recommendation system that combines both techniques.
One application of this algorithm is Netflix.

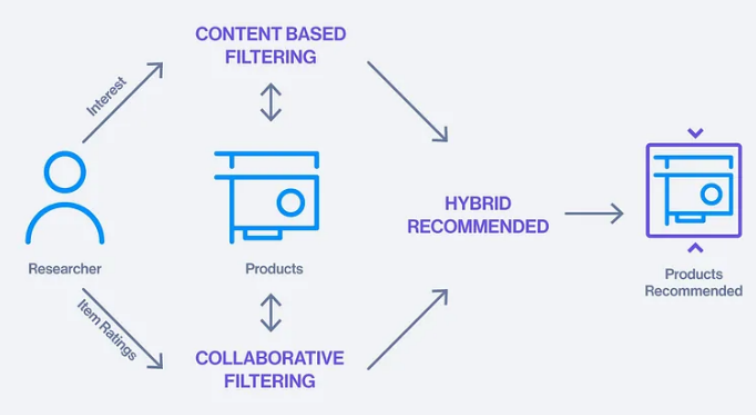

Figure 2.6: Scheme of Hybrid Recommendation Systems

2.4 Challenges

As with any other research field, there are many and various challenges to consider when building a Recommendation System. Some have been there from the start, while others are emerging as the technology improves. This section will introduce two of the more prominent challenges, namely that of cold start and scalability; it will also mention two emerging challenges: privacy and proactive recommendations.

2.4.1 Cold start

A big problem for Recommendation Systems is the so-called cold start problem, which pertains to the sparsity of information [29] about users and items. To adapt to a user, the system needs to know what the user liked in the past. However, when a new user joins the system, nothing is known about the user and, thus, it is impossible to make recommendations.
Systems based on CF are built on community preferences, such as ratings. Consequently, if an item does not have any ratings it will never be recommended to anyone. This problem occurs mainly when the user population is small compared to the item base or it is a new item in the system. It is also problematic to do proper clustering when the amount of ratings in general are low, and bad clustering leads to bad recommendations.
It tends to solve this problem by either getting users to rate items at the start, or by getting them to answer some demographic questions (and then using stereotypes as a starting point, for example, older people like classical music).

2.4.2 Scalability

The biggest problem for scaling a Recommendation System is the amount of operations involved in computing distances. One possible way to solve this is by use of clustering algorithms, or reducing the dimensionality of the data. It is common in Recommendation Systems to have high-dimensional and sparse datasets, so it is essential to be able to reduce the dimensionality of the data. This is necessary for clustering algorithms because of noise in the features, and the computational complexity associated with high dimensionality.
It is also helpful to reduce the dimensions when the feature vectors contain a limited number of values for each object and there is much density between each value. Most of the values will be zero when for example describing the interest relation between an item and a user because most users have just made up their mind about a tiny portion of all

the objects.
Two popular dimensionality reduction methods are Principal Component Analysis (PCA) [14] and Singular Value Decomposition (SVD) [20]. These methods map users and items into a dense and reduced latent space that captures their most prominent features. They provide better recommendations than traditional neighborhood methods [24] as they reduce the level of sparsity and improve scalability [26].

2.4.3 Emerging challenges

Most Recommendation Systems developed so far follow a "pull" model, meaning that the user has to explicitly request recommendations. However, in the modern society where computers are ubiquitous and smartphones are everywhere, it seems natural to imagine that a Recommendation System should be able to detect implicit requests, resulting in proactive Recommendation Systems [42]. The challenge then consists of not only predicting what to recommend, but also when to recommend it.
Another challenge is that of privacy. Privacy and security are increasing concerns regarding Recommendation Systems [27]. The very foundation of the technology is based on knowing as much as possible about the users. To increase the quality of these systems, they collect as much user data as possible. This will clearly have a negative impact on the privacy of the users, and they may start to feel that the system knows too much about them. Therefore, it is important to address this issue in the research community. There is need for systems that sensibly use user data, while ensuring that malicious users cannot get their hands on private data.

2.5 Evaluation Metrics

Once a Recommendation System has been implemented, it is essential to evaluate its performance. However, while accurate predictions are essential, they are insufficient for the deployment of an effective recommendation engine: in applications, users use Recommendation Systems for much more than simply predicting their preferences, and consequently, it is necessary to determine the set of attributes that may influence the performance of the Recommendation System in the context of a given application.
The different properties that can be considered when evaluating a Recommendation System are explained below.

2.5.1 User studies

A user study is conducted by having test subjects perform various tasks that require interaction with the network management system. Subjects are asked questions about their experiences during the tasks and quantitative measurements are collected. Unlike offline evaluation methods, user studies offer the possibility of observing the user in action. Thus, this method does not need to make assumptions about user behaviour and will allow qualitative information on user behaviour to be collected. The disadvantage of this method is that it is very expensive to conduct: in order to obtain a reliable evaluation, subjects have to repeatedly perform more than one task, which entails a high cost.

2.5.2 Online evaluation

Online evaluation, also known as A/B testing, measures the change in user behaviour when the user interacts with different Recommendation Systems under different conditions.
In online evaluation, the system is used by real users performing real tasks. This method can be used to see how the user's behaviour is affected by the different changes in properties made in the evaluation process. Many real systems use an online test system to compare several algorithms.
Typically, such systems redirect a small percentage of traffic to several alternative recommendation engines and record user interactions with the different systems.

2.5.3 Offline evaluation

This type of evaluation is carried out using previously collected user behaviour data, e.g. user ratings. It is assumed that user behaviour after the implementation of the system is the same as the previously collected data set.
The negative factor is that the dataset may need to be narrower and consider aspects of the system as a whole.
In this sense, it can be said that this method is suitable for quickly determining whether the system works well or poorly and at low cost. To carry out these evaluations, it is necessary to simulate the behaviour of a user line in which the user provides feedback on recommendations.

2.5.4 The principal properties of Recommendation Systems

A wide range of properties should be considered when evaluating a Recommendation System. Since different Recommendation Systems try to satisfy different needs, they must be constructed by focusing on various properties, some of which contradict each other.

User preference

This property is the simplest: if a user prefers one system over another, it can be stated that the preferred system is better. This property does not need to be measured and it is easy for the user to have an opinion.

Prediction accuracy

In evaluating prediction accuracy, each catalog entry is given a predicted score. After the user has given these items their true scores, accuracy can be measured by how much the predicted score is lower than true. The root mean square error (RMSE) is the most commonly used parameter. The usage prediction measure shows the amount of successful recommendations.
It is necessary to explain a few terms:

- True Positive(TP): the items recommended to the user that the user classifies as interesting.
- True Negative(TN): the elements not recommended to a user that the user classifies as uninteresting.
- False Positive(FP): the elements recommended to the user that the user classifies as uninteresting.
- False Negative(FN): the items not recommended to the user that the user classifies as interesting.

The simple measure of accuracy is the ratio of correctly predicted instances to the total number of instances, defined by this Equation:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

To obtain a more informative measure of accuracy, precision and recall are used. To get a more informative measure of accuracy, precision and recall is used.
Precision is a measure of the number of successful predicted elements and it is defined by this Equation:

$$precision = \frac{TP}{TP + FP}$$

Recall measures how well the system can recommend items of interest to the user, as defined in the following Equation:

$$recall = \frac{TP}{TP + FN}$$

Coverage

This term can have several meanings, but the most common is the Item Space Coverage. The simplest measure of this is the percentage of all items that may be predicted by the system. Another one is the User Space Coverage. This is the percentage of the users that the system can recommend items to.

Serendipity

Serendipity is the amount of relevant information new to the user through a suggestion. A serendipitous recommendation may lead users to discover interesting things they would not have noticed. It is challenging to define design parameters to quantify serendipity as serendipity measures the distance between suggestions and surprising and attractive items to consumers.

Diversity

Diversity can be defined as the opposite of similarity. In some circumstances, offering a group of comparable items may not be helpful for the user because exploring the item space may take longer. The most often studied method of assessing diversity is the similarity between items, usually based on their content. When diversity increases, the system is less likely to recommend items the user is satisfied with, thus reducing accuracy. The cold start problem is strongly related to the diversity property of a search and selection system.

Robustness

Robustness can be understood as the system's ability to handle false information. Considering how people trust Recommendation Systems, one can influence the system to predict items in their favour: for example, the rating of an item can be increased by making more false users rate it positively. However, it is unrealistic to create a system that is immune to this type of attack. Robustness can be assessed by the system's stability under extreme conditions, such as many requests to the system simultaneously.

Adaptability

This property describes the ability of the system to adapt to rapid changes in the set of elements and their trends. Another type of adaptability is the speed with which user preferences adapt to new user evaluations.

Scalability

It is essential for a Recommendation System to work well on large datasets. Ideally, algorithms should work well on both small and large datasets, but even if they work on small datasets, it does not necessarily mean that they are lawless on many elements. Reducing dimensionality will lead to faster calculation of recommendations on larger datasets, without sacrificing accuracy. The scalability of a Recommendation System is measured by observing the speed and resource consumption when testing the system on data sets of different sizes.

Chapter 3

First Steps towards the Knowledge Graph-based Food Recommendation System

This chapter will provide a comprehensive overview of knowledge graphs and their main applications, focusing on Recommendation Systems. In addition, the issues associated with the development of a Food Recommendation System will be introduced, and the first steps will be taken toward the implementation of a Food Recommendation System based on knowledge graphs that can make recommendations to the user considering both the health aspect and the user's food preferences.
To this purpose, first, the mechanism for constructing a knowledge graph called FoodKG will be introduced, and then its extension to include information associated with the presence of ingredients with specific allergens, the suitability or otherwise of the same ingredients for particular dietary regimes, and their healthiness.

3.1 The Knowledge Graph

This section gives a brief introduction to the history of knowledge graphs, how they are constructed, represented and used.

3.1.1 Brief History of the Knowledge Graph

The term *Graph* refers to a type of data structure, consisting of nodes and edges, suitable for representing relationships between objects.
The Figure 3.1 illustrates the evolutionary process associated with the history of knowledge graphs and related technologies. More precisely, the idea of graph-based knowledge representation can be traced back to the 1960s, when the semantic network was first proposed as a form of knowledge representation. It uses nodes to represent concepts and edges to describe the relationships between concepts in a graph.

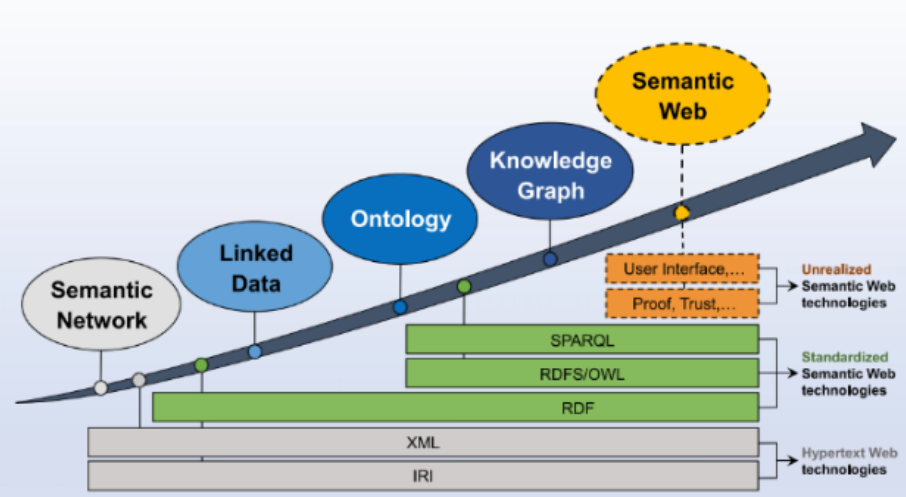

Figure 3.1: The evolution of the knowledge graph

Initially, there was no standard for using node and edge values in semantic networks, and consequently, nodes and edges could be freely defined. However, this led to a number of difficulties in integrating different semantic networks and made the application of semantic networks complex. Subsequently, the standards problem was partially solved through the Resource Description Framework (RDF). It was developed by the World Wide Web Consortium (W3C) as a standard for describing Web resources. Its primary data model is the *subject-predicate-object* triple, which indicates that the two entities, the subject and the object, are linked by a relation, the predicate.
These entities and relationships generally use International Resource Identifiers (IRIs) as indexes in the RDF framework to solve the difficulties of integrating resources from different sources.

Related to RDF, the Semantic web concept, known as Web 3.0, was also introduced. This is a grand idea about the future of the Internet, the ultimate goal of which is that all data on the Internet is published with semantics and is also linked together so that data can be queried, inferred and understood efficiently and intelligently.
In order to build the Semantic Web, W3C helps to build a technology stack called Semantic Web technologies, which could be involved in the construction of the Semantic Web (e.g., RDF). Although Semantic Web remains largely unrealized, these technologies are widely used: Linked data, indeed, is one of its implementations proposed in 2006, which publishes and interlinks datasets on the Internet using Semantic Web technologies. Compared with the semantic network, linked data emphasizes links between Web data and Web resources. However, even after this evolution, RDF still lacked the capacity for abstraction and was unable to describe or distinguish relationships between entities, thus compromising the understanding and inference of knowledge. For this reason, the W3C subsequently proposed the Resource Description Framework Schema (RDFS) and the Web Ontology Language (OWL), i.e. extensions of RDF that add common pre-defined vocabularies to the schema level, so that abstract relationships, such as classes (concepts), instances (objects), subsets and properties, can be represented.
In particular, many data models can be used as schema layers although, the most widely used is the ontology: a specification of knowledge, a formal and explicit description of concepts within a certain domain, properties of each concept and restrictions on facts. The purpose of an ontology is, in fact, to provide a shared understanding of conceptual knowledge and to give a definition to the mutual relations between concepts, making semantic inference possible. RDFS and OWL are the main description languages for ontologies, as they provide good presentation capabilities and semantic support.

The term *knowledge graph*, on the other hand, was proposed in 2012 by Google to describe its knowledge base structured in graphs containing hundreds of millions of entities and relationships. More generally, as can also be seen in the Figure 3.2, a knowledge graph mainly describes real-world entities and their relationships in a graphical representation, and defines the possible classes and relationships of entities using ontology as a schema.
It could be seen as a synonym for a knowledge base, but it has a slight difference: a knowledge graph, when considering its graph structure, can be seen as a graph, and when highlighting its formal semantics, it can be seen as a knowledge base for interpreting and inferring facts. In fact, a knowledge graph is seen as a multi-relational graph of data used to convey real-world knowledge, where nodes represent entities and edges represent different types of relationships. The core of knowledge graphs are instances, while the ontology is often used as a schema and plays a minor role. In general, the number of statements at the instance level of knowledge graphs is much larger than that of the ontology.

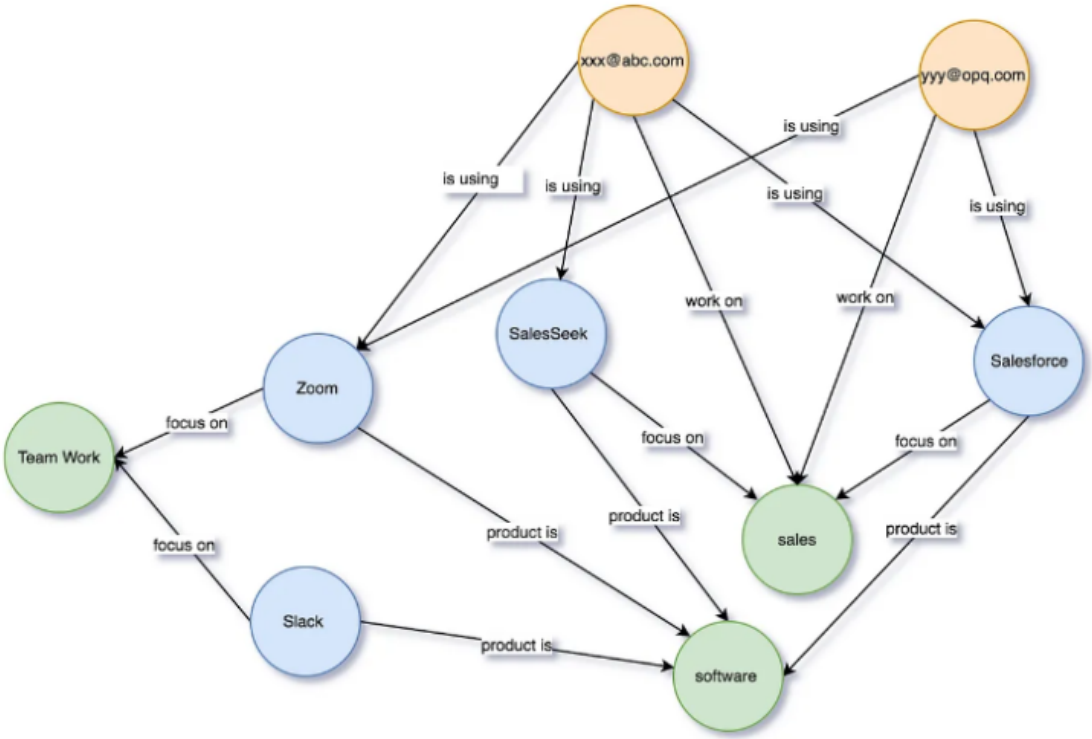

Figure 3.2: An example of knowledge graph

Furthermore, it can be said that a knowledge graph is a graph-structured knowledge base that stores factual information in the form of entity relationships (or literal values), enabling the modeling of real-world entities and their relationships and, consequently, feeding into search engines, natural language understanding systems and, more recently, Recommendation Systems.
To explore knowledge graphs for applications, one must first construct the knowledge graph and, based on the knowledge graph created, an effective representation of knowledge graphs should be needed to support further reasoning and applications, such as research and recommendation.

3.1.2 Knowledge Graph Construction

Completeness, accuracy and data quality are three essential factors that determine the usefulness of knowledge graphs and are influenced by the way in which knowledge graphs are constructed.
Knowledge graphs can be constructed manually or automatically. Manual construction methods include curated and collaborative methods, where the former creates triples from a closed group of experts, while the latter uses an open group of volunteers. Manually constructed knowledge graphs have few or no noisy facts. However, they require a great deal of human effort. Consequently, self-built methods have been explored and have become mainstream. In particular, self-constructed methods can be grouped into two types: The first uses hand-created and learned rules to exploit semi-structured data, such as Wikipedia info-boxes, leading to larger and more accurate knowledge graphs, such as DBpedia. This method can still guarantee high knowledge accuracy. However, semi-structured text still covers a small part of the information stored on the Web and these repositories are still far from being complete.
Consequently, the second approach of extracting facts from unstructured text using Machine Learning and Natural Language Processing techniques is proposed. The knowledge vault is a representative project in this category. In order to reduce the level of noise in the extracted facts, numerous researches have been conducted, which mainly consist of three components:

1. Knowledge extraction which aims to capture entities, attributes and relationships from various data sources. The information is collected and normalized, forming the expression of knowledge. Considering that there are multiple representations for the same entity in many cases and inconsistency of triplets extracted from numerous information sources;

2. Knowledge fusion that is a necessary step. The main processes are entity alignment and entity linking, where entity alignment is the evaluation process to judge whether different entities refer to the same real-world object or not, and entity linking connects entities in the text with corresponding entities in the knowledge graphs;

3. Knowledge refinement which includes various refinement methods, such as entity classification, relationship prediction and anomaly detection, that are used to improve the quality of the structured knowledge graph.

3.1.3 Knowledge Graph Representation and Reasoning

Learning an effective representation for knowledge graphs (i.e., knowledge graph embedding) can encode entities and relations in a continuous low-dimensional vector space. Various methods of learning the representation, such as linear models, neural networks and translation methods. Based on the learned feature representation, knowledge graph reasoning can be further conducted to identify errors and deduce new conclusions from existing data. New relationships between entities can be derived through reasoning and in turn, can be used to enrich knowledge graphs. Various methods of reasoning have been proposed, such as rule-based reasoning and reasoning based on neural networks. Note that neural networks have been widely used for representing and reasoning knowledge graphs due to their powerful non-linear adaptive capacity.

3.1.4 Knowledge Graph Applications

Knowledge graph representation and reasoning can support various tasks, such as relation extraction and entity classification, and real-world applications, such as Question Answering (QA), Information Retrieval (IR) and Recommendation Systems. Here, it have been briefly discussed four critical use cases:

- *Search:* the knowledge graph, in this case, can be used to understand the intent of user queries of the user to support semantic search, which aims not only to find keywords, but also to determine the intent and context of the words used for the query. Semantic search provides more meaningful search results by evaluating the search phrase and finding more relevant results. The knowledge graph enhances semantic search by providing more structured results and better summaries. In this way, the search engine is able to summarize the relevant content of a topic in the form of knowledge graphs, which include the key facts for that particular topic;

- *Question Answering:* has applications in various fields, such as chat-bots. Answering questions using knowledge graphs adds a new dimension to these fields. As L. Hirschman and R. Gaizausk described [25], a QA system based on knowledge graphs answers a natural language question using information stored on the knowledge graph itself. The input question is first translated into a formal query language and then executed on the knowledge graph to obtain the answer. Such systems have been integrated with popular web search engines like Google and Bing and conversational assistants like Siri.

- *Recommendation:* knowledge graph-based Recommendation Systems link users and items, and can integrate multiple data sources to enrich semantic information. Implicit information can be obtained through knowledge graph reasoning techniques to improve the accuracy of recommendations. There are several typical cases of recommendation based on knowledge graphs, such as food, film, and music recommendations;

- *Decision-Making:* the act of choosing between possible solutions to a problem. Knowledge graphs store expert knowledge from different domains to support highly complex decision-making. Knowledge graphs are actively used in the medical domain as one representative domain. When applied to medical knowledge graphs, reasoning on knowledge graphs can help doctors to diagnose disease and control errors to build a decision support system.

3.2 Recommendation Systems Based on Knowledge Graphs

The spread of various networks, such as the Internet of Things (IoT) and mobile networks, databases such as nutritional tables and food composition databases, and social media such as Instagram and Twitter, has led to the generation of enormous amounts of data, which offer researchers an unprecedented opportunity to study various problems and applications, including through data-driven computational methods. However, this data is heterogeneous and multi-source and, therefore, can appear as information silos, sometimes marked by a consequent difficulty in fully exploiting this data.
In this regard, the knowledge graph provides a unified and standardized conceptual terminology in a structured form and, consequently, can effectively organize this data for the benefit of various applications. In the field of food, in particular, the knowledge graph can transform vast amounts of multidisciplinary and heterogeneous food data from multiple sources into a more reusable and digitally connected global Internet of Food for the benefit of science and the food industry itself.
Knowledge Graphs (KG), in fact, play an essential role in organizing information and making it more widely available to both humans and machines. More specifically, the representation and reasoning of knowledge graphs can support various tasks, such as relation extraction and entity classification, but also real-world applications, such as question answering (QA), Information Retrieval and Recommendation Systems. Regarding the latter, in recent years, the integration of a knowledge graph (KG) into Recommendation Systems

as collateral information has aroused great interest and, since through a knowledge graph, it is possible to map items, their attributes, preferences and relationships between users and users, preferences and relationships between users and items, a knowledge graph can, therefore, provide additional information and can be used to suggest potential relationships between items. Knowledge graphs, in fact, can be useful for recommendation in three different respects:

1. The knowledge graph can introduce semantic correlation between items to help find their latent connections and improve the precedence of recommended items;

2. Various types of relationships in the knowledge graph are very useful for extending user interests and increasing the diversity of recommended outcomes;

3. The knowledge graph can bring explainability to Recommendation Systems by connecting historical user data with recommended items.

In particular, there are several cases where knowledge graph-based recommendation is applied, e.g., food recommendation, film recommendation and music recommendation. In the former case, in particular, the elements of recipes and their attributes (i.e., ingredients and categories) mapped in the knowledge graph can enable a food Recommendation System to understand the latent reciprocal relationships between them better and thus make the recommendation results more explainable.
Furthermore, by exploiting such relationships, knowledge graph-based Recommendation Systems are able to provide more accurate and diverse recommendations, while addressing some of the challenges that characterize traditional Recommendation Systems, such as data sparsity, cold-starting and scalability.

More precisely, it is possible to adapt the definition of a Recommender System from the Section 2.2 about adopting a knowledge graph. In fact, in its simplest form, a Recommendation System consists of three consecutive phases: information gathering, learning and recommendation. In this case, the first phase consists of the construction of a weighted graph $G = (U, I, E, w)$, where U, the set of users, and I, the set of elements, are the nodes of the graph and E corresponds to the set of edges. These edges represent the past interactions between users and elements, and the past interactions are described by the function $w: E \rightarrow [0, 1]$.
Next, as in the traditional case, there is the learning phase, in which a Machine Learning algorithm is used to train a model W that approximates w in G.

Finally, in the recommendation phase, the trained model is used to predict, for each possible pair *(u, i)* $\in$ *(U* $\times$ *I)*, the strength of the interaction between user u and object i. From these predictions, it is then possible to derive a list of items that could be recommended to users.

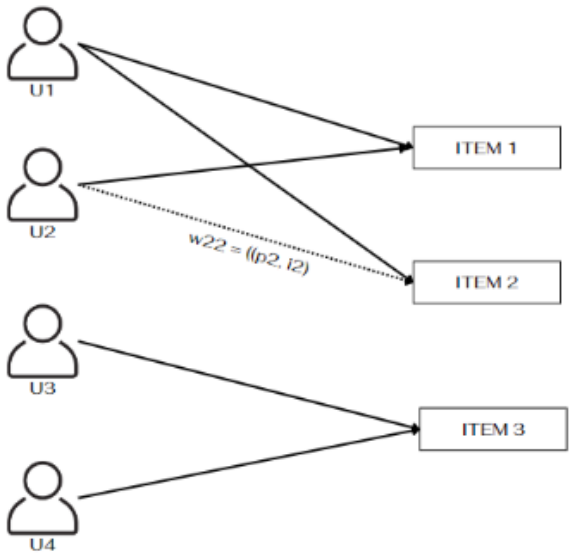

Figure 3.3: The Graph *G = (U, I, E, w)*. The left-hand side contains the set of users *U*, while the right-hand side contains the set of elements *I*. The edges *(E)* connecting sets of users to the elements are the past interactions labelled "interact". The objective of the Recommendation System is to calculate all pairs *(u, i)* in *(U* $\times$ *I)*(e.g. $w_{2,2}$).

Now, It is possible to focus on the importance of using knowledge graphs within Recommendation Systems. As mentioned earlier, the use of knowledge graphs within Recommendation Systems has become widespread in recent years as they overcome the problem of scarcity of user-object interactions and the cold-start problem by exploiting the properties of elements and users and representing them in a single data structure. Knowledge graph-based recommendation systems exploit the graph structure and semantically rich relationships between entities to improve the accuracy and diversity of the Recommendation System and make the recommendation more interpretable. In general, existing KG-based recommendations can be classified into two main categories:

- *Embedding-based methods:* a subclass of knowledge graph-based Recommendation Systems, pre-process a KG using knowledge graph embedding algorithms [16] and

then incorporating the learned entity embeddings into a recommendation framework, as the Figure 3.4 shown. Using knowledge graph embedding algorithms, it is now possible to transform virtually any information into a vector that the system can learn. For example, Zhang et al. [44] propose a two-step approach of first computing embeddings from a knowledge base consisting of structural knowledge, images, and text representing the elements, then using the generated embeddings as input to a CF algorithm. In another research work [45], Zhang et al. use various types of collateral information about items (e.g., review, brand, category, purchases together, ...) to construct a knowledge graph that is subsequently used to build elements and user embeddings. As a second step, the Recommendation System ranks candidate items j in ascending order by a distance between u_i and v_j.
To summarize, many of the embedding-based methods consist of two steps: first, entity embeddings (elements, users, etc.) are learned; second, the embeddings are incorporated into a recommendation learning algorithm.

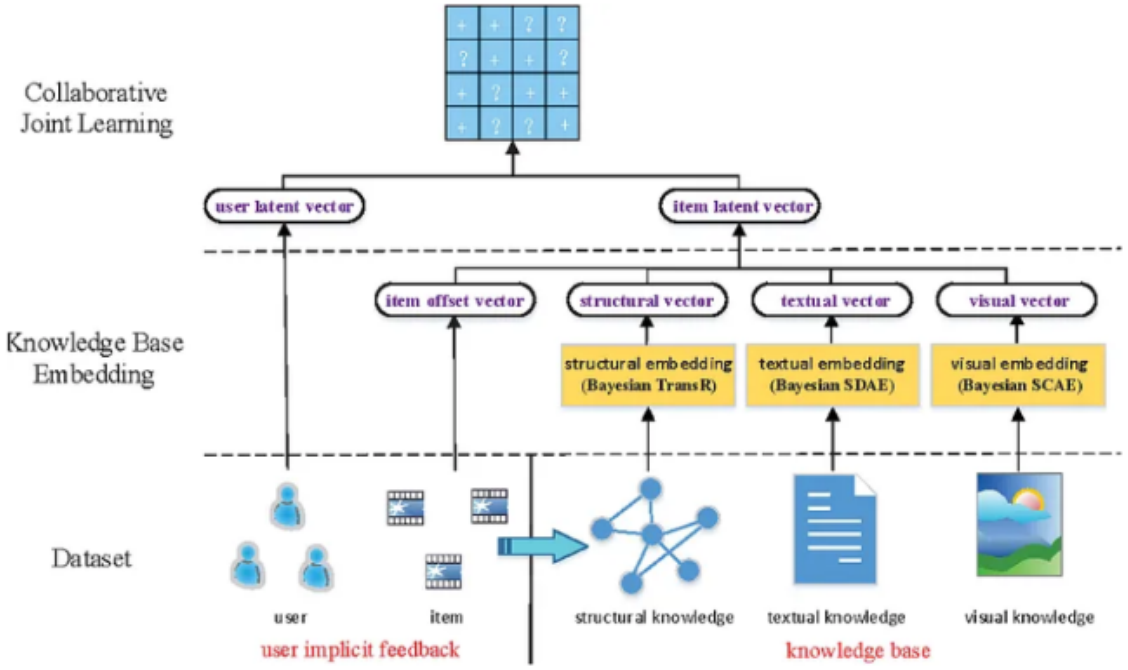

Figure 3.4: Illustration of the Embedding-Based Method

- *Path-based methods:* explore the various patterns of connections between the elements

of a KG to provide further guidance for recommendations. However, they rely heavily on manually designed meta-pathways that are difficult to optimize in practice. For example, Yu et al. [43] use the matrix factorization method to calculate the latent representation of entities for different subgraphs extracted from a heterogeneous KG, then use an aggregation method to group all the generated latent representations to calculate a recommendation probability. Inspired to this previous work, Zhao et al. [46] consider the KG as a heterogeneous information network (HIN). They extract latent path-based features to represent connectivity between users and elements along different types of relationship paths. The disadvantage of these methods is that they generally require specialized knowledge to define the type and number of meta-paths. With the development of Deep Learning algorithms, several models have been proposed to automatically encode KG meta-paths through embedding to overcome the abovementioned limitations.

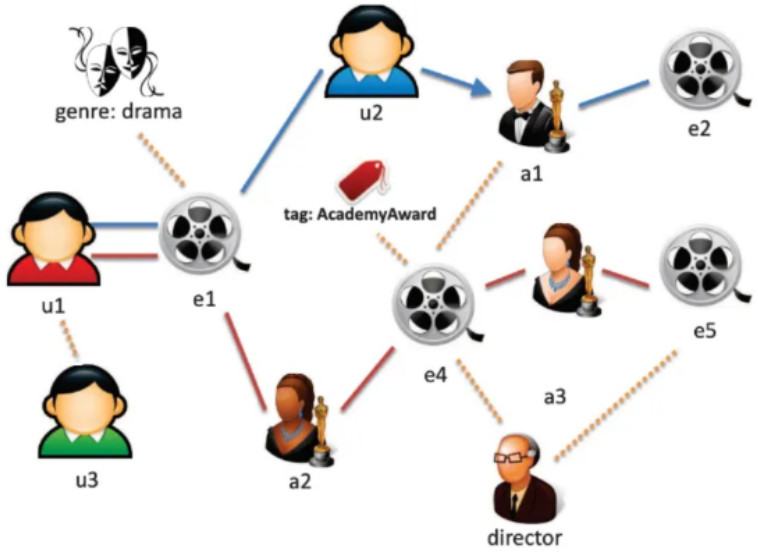

Figure 3.5: User preferences along different meta-pathways

- *RippleNet:* end-to-end framework that naturally incorporates the knowledge graph into Recommendation Systems by considering the entire KG structure rather than exclusively KG triples [39]. It stimulates the propagation of user preferences over the set of knowledge entities by automatically and iteratively extending a user's potential interests along links in the knowledge graph.

In general, knowledge graphs provide a powerful way to structure data for Recommendation Systems by representing entities and their relationships in a graphical format. Several knowledge graph-based Recommendation System methods have been proposed, including path-based, embedding-based and convolutional network-based methods on graphs. These methods offer varying degrees of accuracy and scalability, depending on the size and complexity of the knowledge graph. Although there are obvious challenges in developing these systems, knowledge graph-based Recommendation Systems hold great promise for providing users with more accurate and customized recommendations.

3.3 Food Recommendation System

Online Recommendation Systems have proven helpful in various situations, enabling the user to overcome the problem of information overload, as they assist the user in the decision-making process and, at the same time, can successfully modify their behaviour. Recommendation systems, in fact, have made it possible to disregard the myriad of choices and decisions to be made before arriving at the choice that satisfies an individual's needs and preferences. To this end, they have been successfully employed in a variety of sectors, from film and music recommendations to e-commerce and healthcare.
However, one area that has historically received little attention is food recommendation, especially when compared to other domains of interest such as areas associated with leisure and entertainment. This is surprising, especially when one considers the importance of food for human sustenance, the range of options available, the fact that making food choices is particularly challenging, and that making the wrong choices can entail high personal and social costs. Food, indeed, is central to human life, just think that in ancient times humans faced the task of identifying and gathering food for their survival.
Consequently, food choice is paramount and must satisfy different needs of different individuals, such as basic nutrition, calories, taste, health and social occasions.
Currently, seeking culinary inspiration from authoritative sources is becoming increasingly popular, as are systems that recommend other types of food, such as restaurant meals or supermarket products.
One must, however, also consider the difficulties that distinguish food recommendation from recommendations in other fields: food recommendation, indeed, aims to provide a list of food products classified based on various factors that enable users to meet their needs in a personalized manner. These include, for example, taste preference, perceptual difference, cognitive moderation, culture and even genetic influence.
Consequently, most existing food recommendation methods suggest recipes based on recipe

content, the user's behaviour history (e.g., food history), or dietary preferences.

However, diet, and thus also food recommendations, and other factors such as exercise, sleep, heredity and pollution can play a fundamental role in an individual's health and small adjustments within these can lead to significant results.
Worldwide, lifestyle and diet-related diseases, such as obesity and diabetes, account for 60 percent of all deaths. According to the International Diabetes Federation, some 415 million people suffer from diabetes. This rate is expected to increase further and by 2040, it will be more than 50 percent higher, thus becoming a major threat to global health [38].
Consequently, food computing is emerging as an innovation in this field to alleviate these problems. Health-conscious food Recommendation Systems are often considered an important part of this solution and can encourage users to adopt healthy eating habits.
To this purpose, a recent survey established that most users involved would be willing to adjust taste in favor of healthier recipe options, thus safeguarding their health.
Many reasons, however, make food recommendation challenging, not only in terms of encouraging healthy behaviour, but also in terms of predicting what people would like to eat, as it is a complex, multifaceted, culturally determined process, not only context-dependent.
Furthermore, when developing food Recommendation Systems, there are additional issues to consider, which are not present in other recommendation domains.
Users may have complex and limited needs, such as allergies, or lifestyle preferences, such as the desire to eat only vegan or vegetarian food. Standard approaches work poorly in these cases, and adequate sources for filtering recipes are often unavailable.
Other problems are that food products may have more than one name, ingredients may be prepared in different ways and, unlike recommended products and supports, it is not at all clear whether recommended products can be prepared or consumed due to the potential lack of availability of ingredients, culinary knowledge and equipment.

In this regard, developing a Recommendation System based on knowledge graphs could solve most of these issues. It could also assist the user in recommending recipes and foods suitable for their needs.

3.4 The Food Knowledge Graph FoodKG

There is a vast amount of disconnected food knowledge spanning a whole spectrum of food topics, from nutrients to recipes. In fact, the proliferation of recipes and other food-related information on the Web presents an opportunity to discover and organize diet-related knowledge in a knowledge graph. Several ontologies related to food currently exist, but

they are specialized in specific domains, e.g., agriculture, production, or specific health conditions. There is a lack of a unified knowledge graph that brings together recipes, nutrition, food taxonomies and links to existing ontologies and that is geared towards consumers who want to eat healthily and need an integrated food suggestion service that includes the foods and recipes they encounter daily, together with the provenance of the information they receive.
In this regard, Steven Haussmann et al. proposed FoodKG [23], a food knowledge graph designed to assist food recommendations. This knowledge graph, in particular, is designed to assist individuals in customizing their dietary goals by providing them with information to improve the alignment between their dietary behaviour and nutritional recommendations.
In the following, the procedure of constructing this graph, which will subsequently form the input of the recommendation system, will be described in detail.

3.4.1 Acquisition of FoodKG Data

The Food Knowledge Graph FoodKG is based on three main sources of data:

- *Recipe:* Online recipe sites allow users to browse and share recipes. Some display content from specific commercial sources; others allow users to upload their own recipes. Each website has specific conventions for the presentation of data. In some cases, this includes an effort to provide machine-readable data.
  In particular, there are also large collections of recipe data produced for research and commercial purposes; one example is the Recipe1M dataset [30]. This is, in fact, a recent large structured recipe data corpus comprising more than 1 million recipes and 800 images. It represents the most extensive dataset compared to those already released in this field and includes twice as many recipes and eight times the number of images.
  More precisely, the recipes are taken from over two dozen popular cooking websites and processed through an application that extracted the relevant text from the HTML, downloaded the linked images and assembled everything into a compact JSON schema in which each piece of data was uniquely identified.
  The contents of the Recipe1M dataset can be logically grouped into two levels: The first contains the basic information, including the title, the list of ingredients and the sequence of instructions for preparing the dish, all provided as free text; the second includes all the images associated with the recipe, which are provided as RGB in JPEG format.

- *Nutritional content of ingredients:* The USDA's National Nutrient Database for Standard Reference is used here [5], which is the primary source of food nutrient content data in the United States and provides the basis for most of the food nutrient content databases in the public and private sectors.
  It contains approximately 8,000 records for various food types, up to 150 nutrient components such as protein, carbohydrates, fats, vitamins and minerals, and other crucial nutritional information such as fiber, cholesterol and fatty acid content. This information is based on chemical analysis and average nutritional composition values for each food.

- *Food Ontology to organize ingredients:* Recipe lists and nutrition tables provide information on millions and thousands of entities, respectively, but suffer from a lack of meaning: these components form a solid knowledge graph, but lack an ontology. In order to solve this problem, relevant parts of the FoodOn ontology have been incorporated [18]. FoodOn, in fact, is an open-source and comprehensive food ontology resource, consisting of various facets of the food hierarchy, organizing them by organism of origin, region of origin and so on. This provides connections between related concepts. The Figure 3.6 shows a simplified example of apple products in FoodOn. It describes the relationships between food sources, such as apple and pome plants, different types of food products, such as apple pies and caramelized apples, and related food processes, such as cooking and food coating processes. Moreover, given the tremendous breadth of this taxonomy, it is decided to use only a tiny subset of it: for this purpose, *Ontofox* [40], a tool that extracts terms and axioms from ontologies, is used. In particular, through the use of this tool, it is possible to extract, for each organism, all the children of the food product node, thus capturing a wide variety of food products in a proper hierarchical form and providing a breakdown by organism category, organism group and finally a specific organism of origin.

However, it should be remembered that the collection and integration of data from many sources can lead to several challenges in terms of consistency, accuracy and completeness such as:

- *Invalid data:* some textual data contain illegal characters in an RDF-based knowledge graph, which require escaping. Escaping itself can create problems for entity recognition and resolution; it must be applied consistently at all process stages.

- *Incomplete data:* recipes may lack quantities for ingredients or provide non-standard units of measurement (e.g., "to taste", "as needed", "some shakes"). Nutrient data may be incomplete, with only some nutrients tabulated.

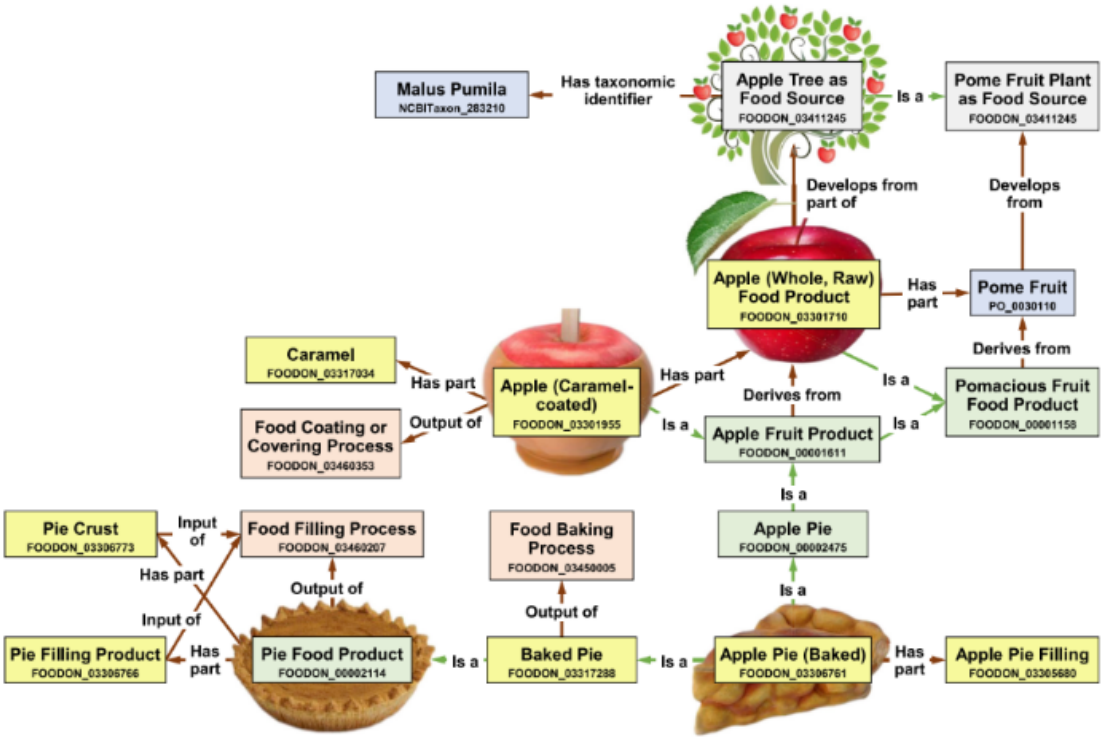

Figure 3.6: A simplified structure of FoodOn

- *Ambiguous entities:* many ingredients are difficult to associate with a specific food. There are many causes, such as local spellings and spelling errors; local names and synonyms and the use of different languages. This can lead to a large number of equivalent names, e.g. corn masa, masa harina, maize flour.
- *Extraneous information:* ingredients are sometimes listed with complicated units of measurement (e.g., 1/3 of a 375 g can of beans) or unnecessary information (e.g., black beans from the shop).

3.4.2 FoodKG Construction

A knowledge graph comprises resources with attributes and entities, relationships between these resources and annotations to express metadata about the resources. FoodKG contains several key components: (i) recipes and their ingredients, (ii) nutritional data of individual foods, (iii) additional food knowledge and (iv) links between these concepts.

Recipes

Each recipe describes the ingredients needed to produce a dish. It consists of a unique identifier, accompanied by its name, any labels provided and a set of ingredients. Each ingredient's name, unit of measurement and quantity are indicated. An example of the resulting structure is shown in Figure 3.7.

The individual ingredient entries usually appear in the form (quantity, unit, name), such

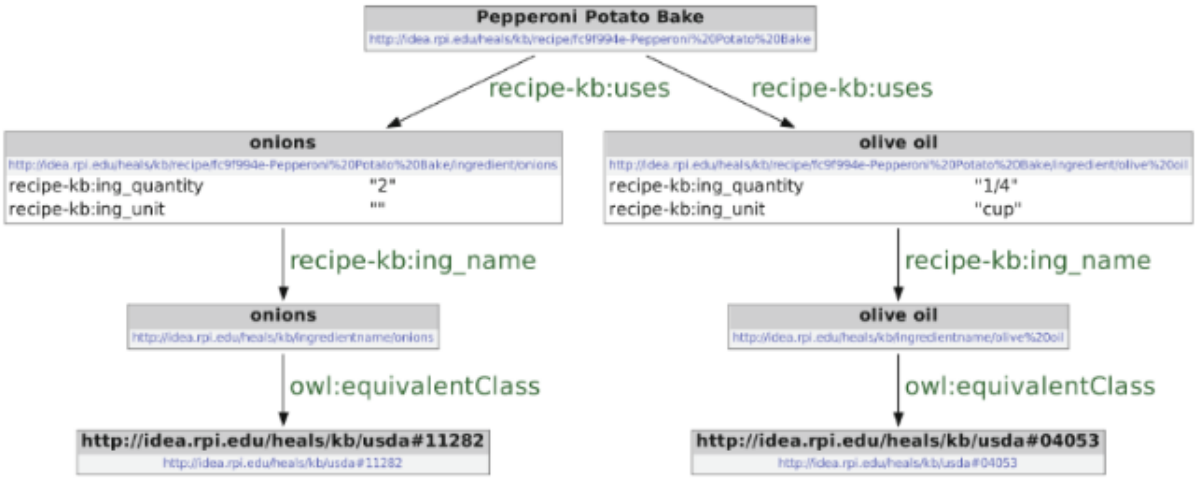

Figure 3.7: Example of an imported recipe, pruned to show only two ingredients

as 2 cups of flour or 1 1/2 lb of chopped cabbage. Due to the lack of context, parsing these sentences with natural language processors is complex and naive parsing methods fail due to minor quirks.

In order to parse these records effectively, the following steps were used: (1) Statements with parentheses, such as (just picked up) or (or chicken), are deleted. These provide additional information to the reader, but are optional to understand the recipe's components. Similarly, text following the first comma is removed, since it generally describes additional qualities for the ingredient. Even if these have a meaning, they are less significant than the name itself. (2) Numerical values are removed and saved as quantities. (3) A list of units is compared with the first word of the string; if one matches, it is removed and stored as a unit. (4) The remaining text is tokenized with the Natural Language Toolkit. Adjectives that are not descriptive of color are removed (e.g., green peppers, red onions), while descriptors are removed, as are verbs and adverbs and text that follows a conjunction. Finally, Nltk's *WordNetLemmatizer* eliminates plurals and the resulting text is saved as a

name.
Examples of input and results are shown in Table 3.1.

Input	Quantity	Unit	Name
1 cup milk	1	Cup	Milk
1 tablespoon parsley, chopped	1	Tablespoon	Parsley
6 tablespoons red currant jelly	6	Tablespoons	Red currant jelly
1 cup butter, softened	1	Cup	Butter

Table 3.1: Examples of processed ingredient data

Nutrients

It is possible to produce a network of foods and their ingredients from recipes. However, without information on the nutritional content of each ingredient, no meaningful health suggestion can be made. To attach this information, the USDA dataset is used. The USDA dataset is in fact, a tabular database describing several dozen nutritional statistics, such as calories, macro-nutrients (protein, carbohydrates, fat) and micro-nutrients (vitamins and minerals).
The nutrients are provided per 100 grams of food. Two non-massive measurements of the food are also provided, together with the number of grams in each measure.
Here, the Semantic Data Dictionary approach [34] is adopted to convert the tabular data into a form that can be integrated into FoodKG, which produces RDF triples from non-triple data sources.

The Semantic Data Dictionary (SDD) specification enables the extension and integration of data from multiple domains using a common metadata standard. The Semantic Data Dictionary, indeed, is a specification that formalizes how to assign a semantic representation to data by annotating dataset variables and their values using concepts drawn from best-practice vocabularies and ontologies. It is a collection of individual documents, each of which plays a role in the creation of a concise and coherent knowledge representation, including the dictionary, code, time sequence and code mapping specifications and the information sheet, which is used to link these elements of the semantic data dictionary together.
Some examples of data converted into knowledge graph concepts are given in Table 3.2.

Given these data, it is possible to define the shape of the resulting knowledge graph through semantic relations, as can be seen in Table 3.8, where *Column* represents the

Id	Description	Water	Energy	Protein	Lipid	Carbohydrate
1001	Butter, with salt	15.87	717	0.85	81.11	0.06
1009	Cheese, cheddar	37.1	406	24.04	33.82	1.33

Table 3.2: Examples USDA data

column in the raw data, *Attribute/Entity* represents the rdf:type of that nutrient, *Unit* refers to the unit of measurement for that nutrient from community-accepted terminologies such as DBpedia and the Unit Ontology and *Label* provides a textual description for the data element that can be used in text mining and autocompletion activities in applications using the FoodKG.
In particular, the various prefixes in the annotations in Table 3.8 point to the following ontologies:

- chebi: Chemical Entities of Biological Interest Ontology (https://www.ebi.ac.uk/chebi);
- dbr: DBpedia Resource Ontology (http://dbpedia.org/resource/);
- sio: Semanticscience Integrated Ontology (http://semanticscience.org/resource/);
- envo: Environment Ontology (http://purl.obolibrary.org/obo/envo.owl);
- foodon: Food Ontology (http://purl.obolibrary.org/obo/foodon.owl);
- schema: Schema.org mappings (https://schema.org/);
- uo: Ontology of units (http://purl.obolibrary.org/obo/uo.owl).

After completing these annotations, the semantic data dictionary conversion script is run to convert the USDA tabular data into quadruples, which are triples grouped into named graphs.

3.4.3 FoodKG Augmentation

Now, with all the data imported, it has been built an isolated dataset. Therefore, the second step in constructing the knowledge graph is linking. In this regard, it is necessary to utilize various entity resolution techniques to link the multiple concepts together automatically. To ensure that the dataset can be expanded and updated practically, well-designed linked data techniques have also been adopted to establish the origin of these derived relationships:

Column	Attribute/entity	Unit	Label
Id	chebi:33290, dbr:Food		USDA Id for the food
Description	sio:StatusDescriptor		Short description
Water	envo:00002006, chebi:15377, dbr:Water	dbr:Gram, uo:0000021	Water (g)
Energy	foodon:03510045	dbr:Kcal	Energy (Kcal)
Protein	dbr:Protein	dbr:Gram, uo:0000021	Protein (g)
Lipid	dbr:Lipid	dbr:Gram, uo:0000021	Lipid Total (g)
Carbohydrate	dbr:Carbohydrate, schema:carbohydrateContent	dbr:Gram, uo:0000021	Carbohydrate (g)
Sugar	dbr:Sugar	dbr:Gram, uo:0000021	Sugar Total (g)
Calcium	dbr:Calcium	uo:0000022	Calcium (mg)

Figure 3.8: Semantic structural representation of a subset of the USDA data

- *Entity resolution:* Names are the most apparent shared attributes between the recipe, nutrient and food domains. For this reason, entity resolution techniques that work on strings, such as cosine similarity, have been widely adopted.

- *Selection of entities:* It has been advantageous to limit the domain of concepts to be compared, both for performance reasons (matching is linearly expensive concerning the number of entities) and to maximise accuracy (more spurious entities to be compared cause more false positives). Strictly how this is done depends on the datasets being compared. For example, many food categories are rarely seen as ingredients but, crucially, have similar names to relevant food types. The USDA's standard reference list contains a large number of entries for baby food, with names such as "Baby food, apple juice" and "Baby food, lamb, strained" being removed, as they cause problems in linking ingredients.
  Categories such as "fast food" and "sweets" are similarly removed. For example, "brown sugar" is put together with candies and jellies, but it is desirable to keep it in FoodKG. Text beyond the third comma is also ignored, as it is found that the distinction between entities becomes meaningless at that point and this also speeds up the linking process.
  Other data sources are much more extensive; for example, linking with the DBpedia knowledge graph is tested. Unfortunately, many entities in the DBpedia dataset have incomplete or inaccurate typing; non-food items have the type Food, while many edible items do not. Therefore, it is decided to use heuristics to select potential ingredients. All DBpedia resources marked as "ingredientOf" and everything with a

carbohydrate value are included. This tended to produce a subset of real foods and, while losing some entities, eliminated a large number of incorrect choices.

- *Provenance and publication information:* To provide a clear provenance for every claim made in FoodKG, including imported knowledge and inferred links, extensive and consistent use has been made of the RDF Nanopublication specification [21]. Nanopublications, in fact, represent atomic units of publishable information, attaching information about their provenance and who/what published it. They express this knowledge through linked data, using four named graphs:
  - The assertion graph contains the statements made. For a recipe, this includes the title, tags (if present) and ingredients. Ingredients are described by their name, units and quantity;
  - The provenance graph contains information on where the claims are derived from. For a recipe, this could be the URL from which the data is retrieved, or any other reference that refers to the original data;
  - The publication information graph explains who created the nanopublication;
  - The head graph links the three previous graphs, making searching for the three components possible.

Through these steps, it is possible to output the FoodKG knowledge graph which consists of three serialized RDF files, in trig format, (i) usda-links.trig, (ii) foodon-links.trig, (iii) foodkgcore.trig and, as also shown in the Figure 3.9, spans over 67 million triples obtained by adding all the triples that make up the USDA, Recipe1M and FoodOn KG subsets and the links between them.

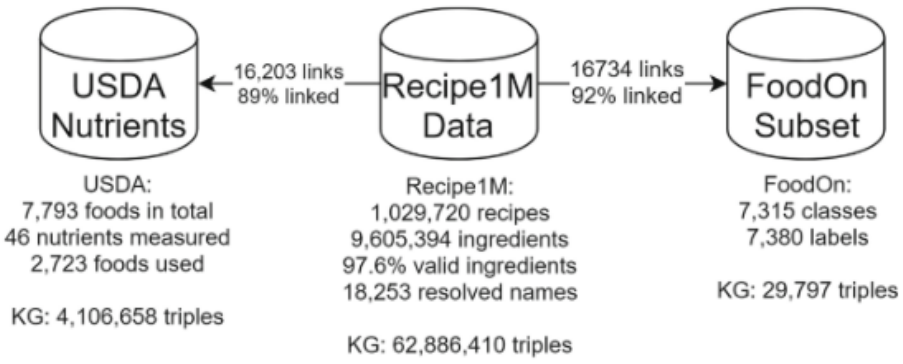

Figure 3.9: An overview of the Food Knowledge Graph (FoodKG)

The Figure 3.10 illustrates the schema of the structure of the newly constructed FoodKG.

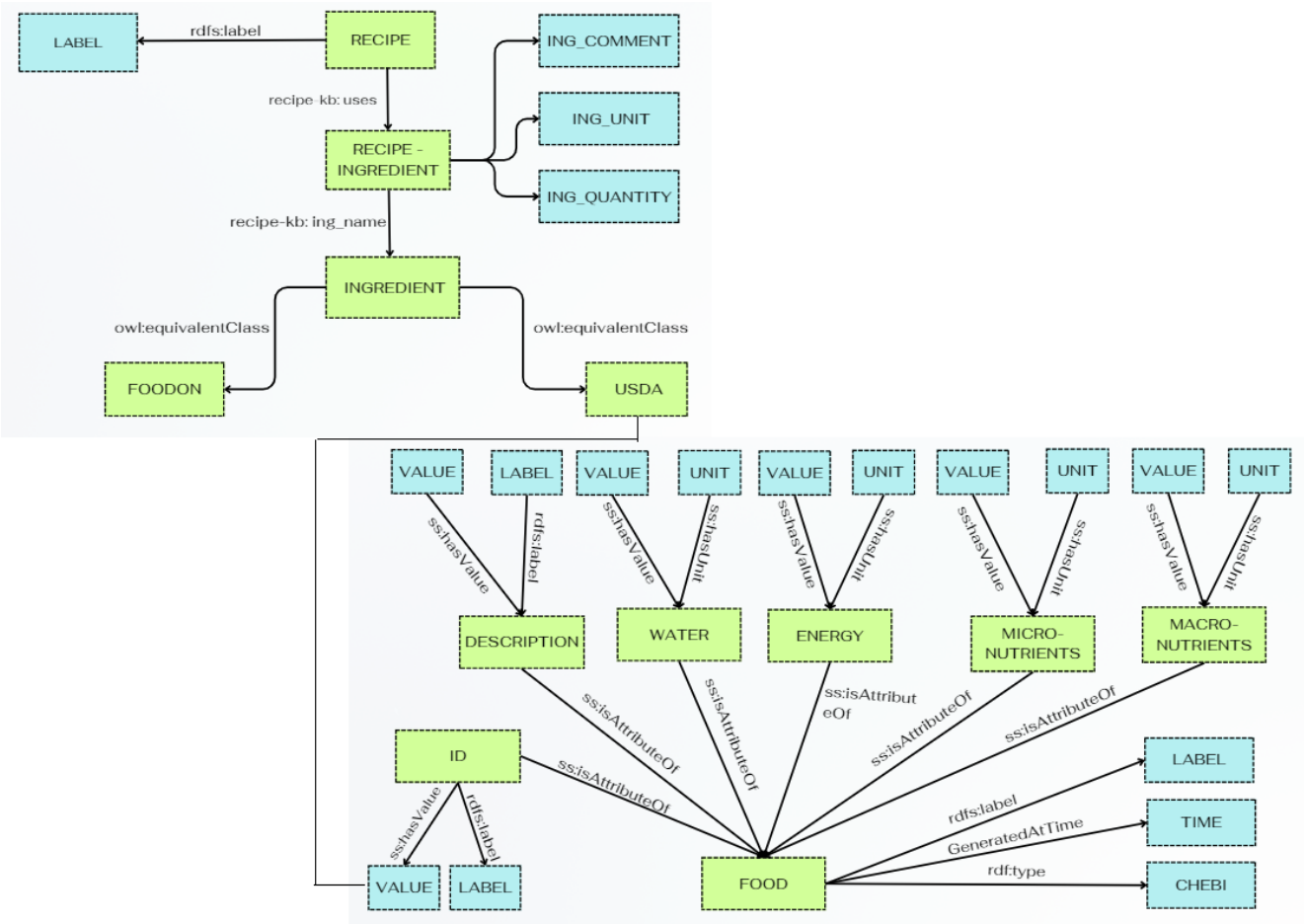

Figure 3.10: FoodKG Structure

3.4.4 FoodKG Extension

The proliferation of recipes and other food information on the Web represents an opportunity to discover and organize diet-related knowledge: the knowledge graph created, FoodKG, is specifically oriented towards consumers who wish to eat healthily and who need an integrated food recommendation service, which includes the foods and recipes they encounter daily, together with the provenance of the information received.
In fact, it can be seen as the input of a recipe Recommendation System aimed at creating personalized recommendations that can promote healthier lifestyles, considering both the health aspect and the user's food preferences.
In this regard, the following section will describe in more detail the steps involved in the manipulation of the newly created FoodKG, with the idea of attaching to it information regarding the presence of ingredients with specific allergens, the adaptability or otherwise of the same ingredients to particular dietary regimes and their healthiness.
The following chapters will describe an intelligent and automated method for recommending food that is capable of adapting to the user's dietary needs and favoring the recommendation of healthier recipes.

First of all, due to the considerable size of the three output files (foodkgcore.trig exceeds 5 GB), it is necessary to store FoodKG on Blazegraph [1] instead of importing it directly into Python via the RDFlib library.
Blazegraph, indeed, is an open-source, high-performance graph database designed to store and query large-scale graphs efficiently. It is a NoSQL database system specialized in handling graph data, a data structure representing relationships between entities such as nodes and edges in a graph.
In fact, it is often used as a triple store to storage and query Resource Description Framework (RDF) data. It also supports the SPARQL query language, designed specifically for querying RDF data, which allows complex queries to retrieve and manipulate graph data.
It scales horizontally to handle large data sets and high query loads, consequently, it can efficiently handle large graphs using various optimization techniques to ensure fast query processing, making it suitable for applications with high-performance requirements.
Due to its high performance, it is suitable for applications where modeling and querying of graph data are crucial, such as semantic web applications, knowledge graphs and data integration scenarios.
In addition, Blazegraph provides a RESTful web services API that allows developers to interact with the database programmatically, facilitating the integration of Blazegraph into various applications.

In particular, since Blazegraph supports Pymantic, a Semantic and RDF Web Library for Python, it is possible to establish the connection to the SPARQL endpoint relating to the storage of FoodKG, thus allowing the data stored in that SPARQL endpoint to be queried and manipulated using Python.

In the following sections, the most common food allergies and diets in recent years will be discussed in more detail, as well as the calculation of the healthiness of ingredients and recipes and the subsequent entry of information about them into FoodKG.

The inclusion of food allergy and diet information

A food allergy is a reaction of the immune system to a food or food component. Food allergy sufferers are forced to systematically eliminate the allergen involved in their diet. In collaboration with the European Commission, the FAO (Food and Agriculture Organization) [17] has compiled a list of the most allergenic foods. In fact, 90% of all food-based allergic reactions are mainly caused by eight foods:

1. *Egg allergy:* the egg contains many proteins with a potential allergic effect. Of these, the main ones are ovomucoid, ovalbumin and ovotransferrin; only the last two are heat labile and denature with cooking, losing their ability to produce allergic manifestations. Knowing to which protein one is sensitized is, therefore, essential to set up an appropriate exclusion diet. In particular, eggs are used in the preparation of most cakes, mousses, sauces, pasta, and quiches, but also in some meat products and mayonnaise and sometimes as a finishing touch in some desserts.

2. *Milk allergy:* allergy to cow's milk protein is the leading cause of food allergy and affects children in particular. It can give rise to even very severe reactions, even through the simple inhalation of milk powder particles. Cow's milk is present in yogurt, cream, cheese, butter, milk powder and foods glazed with milk.

3. *Peanut allergy:* until a few years ago, peanut allergy was a problem restricted to the USA; now it is spreading to Europe and becoming a significant allergy in children. An allergic response to peanuts can occur within minutes of exposure. Peanuts are often found in sauces, cakes, desserts, peanut oil, peanut flour and peanut butter.

4. *Nut allergy:* this category includes walnuts, almonds, pecans, hazelnuts, Brazil nuts, cashews, pistachios, pine nuts, macadamia nuts and chestnuts. It is mainly children

who have a primary allergy to nuts (almonds, hazelnuts or walnuts). Unlike other allergies, this often does not disappear as they grow up. People with a primary nut allergy mostly react to so-called reserve proteins. Different in each type of nut, they have in common, however, that they are destroyed neither by heat nor by gastric acid. Nuts can therefore trigger allergic reactions in raw, roasted and cooked form. These foods can often be found in: sauces, desserts, crackers, bread, ice cream, marzipan, grains (e.g. ground almonds), and oils.

5. *Fish allergy:* it products is one of the most frequent food allergies in Scandinavian countries, where cod allergies are "all the rage". On the possible occupational origin of the disease, the same applies as for milk.

6. *Shellfish and Crustaceans allergy:* it is an adverse immune reaction to specific proteins contained in these foods. One of the allergenic proteins identified is tropomyosin, which is resistant to heat and cooking. Therefore, those allergic to seafood must be careful with raw and cooked products, as with fish. Allergy to shellfish and crustaceans is caused by sensitization to proteins totally different from fish. Most people with shellfish and crustacean allergies can therefore consume fish, as long as there has been no contact with crustaceans and/or mollusks. In some patients, both allergies can be present, but it is linked to a double sensitization. Therefore, lobster, shrimps, prawns, crab, langoustine, granseola should be eliminated from the diet as far as crustaceans are concerned, and squid, octopus, cuttlefish, mussels or mussels, oysters, sea truffles, clams, tellins, cannolicchi and land snails as far as mollusks are concerned.

7. *Wheat allergy:* it is an allergic reaction that occurs following the consumption of food containing cereal; as in the case of other forms of allergy, it is an activation of the immune system, which becomes sensitized to certain substances mistaken as a threat to the body's health. The reaction is usually triggered by food containing wheat, but in some cases, inhaling the flour is enough to trigger the symptoms (contact allergy). It is not yet clear why some people develop food allergies, but it is thought that the cause may be a combination of genetic predisposition and factors. Avoiding wheat is the primary prevention and treatment strategy, which is unfortunately made difficult because the allergen is often contained in different types of food.

8. *Soy allergy:* soy is another highly allergenic food; ingredients such as tofu, soy flour, soy milk and soy protein are made from soy. It should also not be forgotten that some components of soy, such as lecithin and protein hydrolysates, are widely used

as food additives and can, therefore make it dangerous to eat the foods to which they are added.

In addition to increasingly widespread allergies, adopting a diet without consuming meat and fish has also begun to spread in recent years. In fact, numerous epidemiological studies investigate the distribution and frequency of diseases or other events in a population and show that a diet based mainly on the consumption of plant-based foods is fundamental to maintaining health. Furthermore, according to a Position Paper of the Academy of Nutrition and Dietetics [31], vegetarian diets, if properly planned, can promote health by reducing the risk of developing certain medical conditions, including ischemic heart disease, type 2 diabetes, hypertension, certain cancers, and obesity. Added to this is that everyday food choices can also influence the planet's well-being.
In particular, among the diets that exclude meat and fish today, the vegetarian and vegan diets are increasingly used: The former, indeed, bans the use of meat and fish (including shellfish and crustaceans) but allows the use of "animal-derived" foods such as eggs, milk, cheese and honey; the latter, on the other hand, does not include any food of animal origin (meat, fish, milk and dairy products, eggs and honey) and is based on the consumption of cereals, pulses, vegetables and fruit, both fresh and dried, vegetable oils, vegetable drinks and seeds.

Consequently, a process of analysis and classification of its ingredients is carried out on FoodKG in order to incorporate into it information on whether an ingredient may or may not be suitable for people with a specific food allergy, among those described above, or who have embarked on a particular food, vegetarian or vegan diet.
First of all, it is necessary to obtain all ingredients in the FoodKG by means of the SPARQL query, shown in the Figure 3.11:

```
SELECT DISTINCT ?ingredient ?ingredient_name
WHERE {?ingredient rdf:type <http://idea.rpi.edu/heals/kb/ingredientname> .
       ?ingredient rdfs:label ?ingredient_name.
      }
```

Figure 3.11: FoodKG Ingredients extraction

Subsequently, a meticulous analysis is carried out on the ingredients derived from which it can be asserted that FoodKG consists of a wide variety of ingredients, some dictated by a single food component, others derived from the combination of several food components. There are, for example, products such as "chicken", "mango", "lime", "mozzarella cheese" "and olive oil" that are easily classified as food allergies and vegetarian or vegan diets.

Still, there also appear other products such as "hot dog", "pancake", "biscuit", "crepe", "macaroon" that, are also used as basic ingredients for a given recipe but, in reality, are, in turn, made up of other ingredients.
Consequently, it is impossible to make FoodKG's ingredient classification process fully automatic. Still, a lengthy semi-automatic mechanism is adopted to check the composition of each ingredient present and, based on this, to classify it correctly. Specifically, the following labels are associated with each ingredient:

- *All_milk* and *Adapt for allergy to milk or Not for allergy to milk:* to indicate whether or not a certain ingredient would be suitable for people allergic to milk;
- *All_eggs* and *Adapt for allergy to eggs or Not for allergy to eggs:* to indicate whether or not a certain ingredient would be suitable for people allergic to eggs;
- *All_peanuts* and *Adapt for allergy to peanuts or Not for allergy to peanuts:* to indicate whether or not a certain ingredient would be suitable for people allergic to peanuts;
- *All_nuts* and *Adapt for allergy to nuts or Not for allergy to nuts:* to indicate whether or not a certain ingredient would be suitable for people allergic to nuts;
- *All_soya* and *Adapt for allergy to soya or Not for allergy to soya:* to indicate whether or not a certain ingredient would be suitable for people allergic to soya;
- *All_wheat* and *Adapt for allergy to wheat or Not for allergy to wheat:* to indicate whether or not a certain ingredient would be suitable for people allergic to wheat;
- *All_fish* and *Adapt for allergy to fish or Not for allergy to fish:* to indicate whether or not a certain ingredient would be suitable for people allergic to fish;
- *All_shellfish* and *Adapt for allergy to shellfish or Not for allergy to shellfish:* to indicate whether or not a certain ingredient would be suitable for people allergic to shellfish;
- *Vegetarian* and *Adapt for Vegetarian or Not for Vegetarian:* to indicate whether or not a particular ingredient would be suitable for people who have adopted a vegetarian diet;
- *Vegan* and *Adapt for Vegan or Not for Vegan:* to indicate whether or not a particular ingredient would be suitable for people who have adopted a vegan diet.

Subsequently, a SPARQL query is generated for each label type, an example of which is shown in the Figure 3.12, by means of which the result of the classification could be stored within FoodKG.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
INSERT DATA {
    ingredient_name rdfs:milkAllergy all_milk .
}
```

Figure 3.12: Example of how to add dietary labels into FoodKG

The inclusion of health information

An internationally recognized standard is used to incorporate the healthiness of ingredients and recipes into FoodKG: the "traffic light" system of the UK Food Standards Agency (FSA) [33] for food labeling, the UK government department set up in 2000 and responsible for ensuring that food products sold are safe to consume and accurately labeled to safeguard public health.
The approach adopted by the FSA standard, shown in the Figure 3.13, consists of a labeling system that ranks the four main macro-nutrients, sugar, sodium, fat and saturated fat, on a traffic light scale using information per 100 grams or 100 ml of food: green to indicate that the macro-nutrient is in the range defined as healthy, amber if it is in an intermediate range and red if it is unhealthy.

Text	LOW[8]	MEDIUM	HIGH	
Colour code	Green	Amber	Red	
			>25% of RIs	>30% of RIs
Fat	≤ 3.0g/100g	> 3.0g to ≤ 17.5g/100g	> 17.5g/100g	> 21g/portion
Saturates	≤ 1.5g/100g	> 1.5g to ≤ 5.0g/100g	> 5.0g/100g	> 6.0g/portion
(Total) Sugars	≤ 5.0g/100g	> 5.0g to ≤ 22.5g /100g	> 22.5g/100g	> 27g/portion
Salt	≤ 0.3g/100g	> 0.3g to ≤ 1.5g/100g	>1.5g/100g	>1.8g/portion

Figure 3.13: The FSA Standard "traffic light" system

To this purpose, first of all, the SPARQL query expressed in the Figure 3.14 is formulated in such a way as to have a link between all the ingredients in the FoodKG with the relevant four macro-nutrients described in the FSA table needed to calculate the healthiness index.

```
PREFIX recipe-kb: <http://idea.rpi.edu/heals/kb/>
   PREFIX ss: <http://semanticscience.org/resource/>
   SELECT DISTINCT ?ingredient ?sugar_value ?sodium_value ?fat_mono_value ?fat_poly_value ?fat_sat_value
   WHERE {
        ?ingredient rdf:type <http://idea.rpi.edu/heals/kb/ingredientname> .
        ?ingredient rdfs:label ?label .
        ?ingredient owl:equivalentClass ?id .
        ?id ss:isAttributeOf ?food .

        ?sugar ss:isAttributeOf ?food .
        ?sugar rdf:type <http://idea.rpi.edu/heals/kb/usda#sugar> .
        ?sugar ss:hasValue ?sugar_value .

        ?sodium ss:isAttributeOf ?food .
        ?sodium rdf:type <http://idea.rpi.edu/heals/kb/usda#sodium> .
        ?sodium ss:hasValue ?sodium_value .

        ?fat_mono ss:isAttributeOf ?food .
        ?fat_mono rdf:type <http://idea.rpi.edu/heals/kb/usda#fat_mono> .
        ?fat_mono ss:hasValue ?fat_mono_value .

       ?fat_poly ss:isAttributeOf ?food .
       ?fat_poly rdf:type <http://idea.rpi.edu/heals/kb/usda#fat_poly> .
       ?fat_poly ss:hasValue ?fat_poly_value .

       ?fat_sat ss:isAttributeOf ?food .
       ?fat_sat rdf:type <http://idea.rpi.edu/heals/kb/usda#fat_sat> .
       ?fat_sat ss:hasValue ?fat_sat_value .

   }
```

Figure 3.14: The SPARQL query to extract Ingredients and their macro-nutrients extraction

Subsequently, to derive a single metric, the procedure of Sacks et al. [35] is followed, which first assigns an integer value to each color, *green = 1, amber = 2, and red = 3*, and then sums the scores for each macronutrient, resulting in a final value ranging from 4, in the case of a particularly healthy recipe to 12, in the case of a very unhealthy recipe.
Specifically, after obtaining for each macronutrient that made up an ingredient an integer value concerning its reference range in FSA and summing up the results for the four macro-nutrients, it is then possible to attribute to each ingredient an "ingredient health value" label indicating the degree of healthiness of that particular ingredient and, employing the SPARQL query shown in the Figure 3.15, it is possible to store this value within the FoodKG.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
INSERT DATA {
    ingredient_name rdfs:ingredient_health_value ing_health_value.
}
```

Figure 3.15: Example of how to add healthy labels into FoodKG

A similar procedure as above is also performed for recipes; in this case, the average of the "ingredient health values" of all the ingredients in the recipe is taken into account to calculate the "recipe health value."

3.4.5 FoodKG Final Structure

Below in the Figure 3.16, the structure of the FoodKG is shown, after having undergone the extension mechanism: recipes, indeed, have a new associated label related to the health degree attributed to them; ingredients, on the other hand, in addition to the label related to their health degree, have also associated labels related to whether or not they are suitable for a particular food allergy, among the most common ones, and for a particular food diet, vegetarian and vegan.

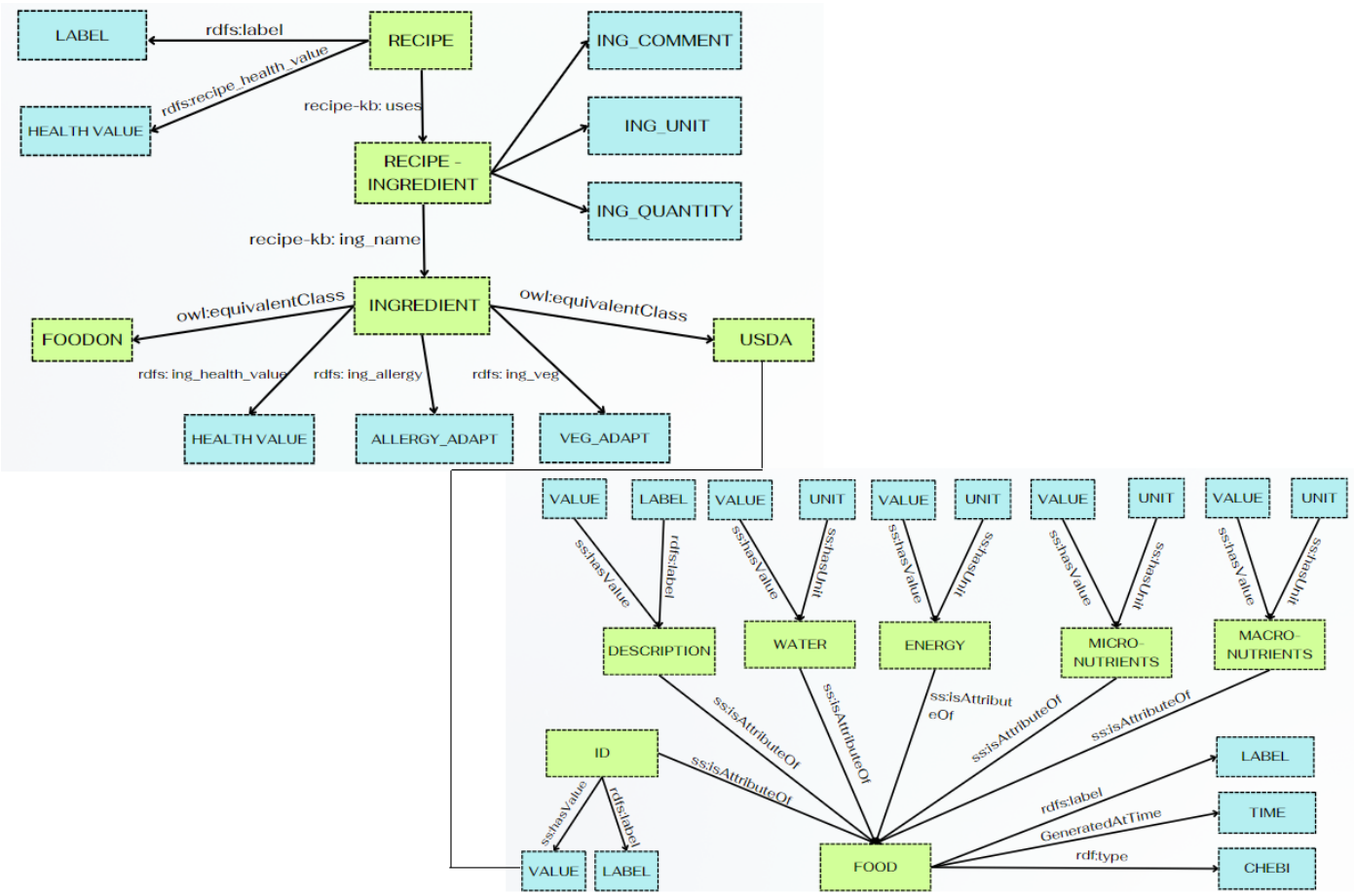

Figure 3.16: New FoodKG Structure

Chapter 4

The Healthy Food Recommendation System over Semantic-Driven Knowledge Graph

This chapter describes the design and development phase of the Food Recommendation System based on the FoodKG, aimed at providing the user with personalized recommendations considering both the health aspect and the user's food preferences.
The first part reports on the applicability of the main categories of Recommendation Systems to the context under analysis, intending to identify the most appropriate category. In this regard, the pipeline of the proposed Recommendation System is also briefly introduced.
The second part, on the other hand, describes the proposed solution for the realization of such a Recommendation System, starting with extracting a FoodKG subgraph to the steps associated with developing the Recommendation System learning model and generating recipe recommendations.

4.1 Choice of Recommendation System Type

This section describes the evaluations of applicability to the context under analysis of the main categories of Recommendation Systems presented in Chapter 2, intending to identify the most appropriate approach to provide the end user with personalized recommendations considering both the health aspect and the user's food preferences.
Regarding Collaborative Filtering Recommendation Systems, they rely exclusively on user

evaluations to provide recommendations. However, in the newly created FoodKG, there are no user evaluations; therefore, this Recommendation System is not applicable. If, on the other hand, Content-based Recommendation Systems are considered, they are based on the description of items and the history of user interactions with them. In this case, the description of the items could be defined based on the nodes and edges in the FoodKG, but there would be no information on the user's interactions with the recipes.
Based on this, the idea is to develop a Recommendation System that relates to an application whose purpose is to interface with the user in a front-end and back-end manner so that personal information can be derived from it.
In this way, as can also be seen in the pipeline shown in Figure 4.1, the user would be asked, during registration, to indicate whether they have specific food allergies and whether they are following a vegetarian or vegan diet. At this point, this information is used for the generation of a FoodKG subgraph so that it can only contain recipes suitable for that particular user and from the subset created, with the help of the Recipe1M dataset, ten randomly selected recipes are shown to the user, of which the user must choose five.
The five recipes selected by the user constitute their initial preferences in the development of the Recommendation System, while the remaining ones constitute the non-preferences. In this way, the Recommendation System can be defined, which recommends to the user in question a healthy recipe suitable, first of all, for their dietary restrictions but also their preferences.

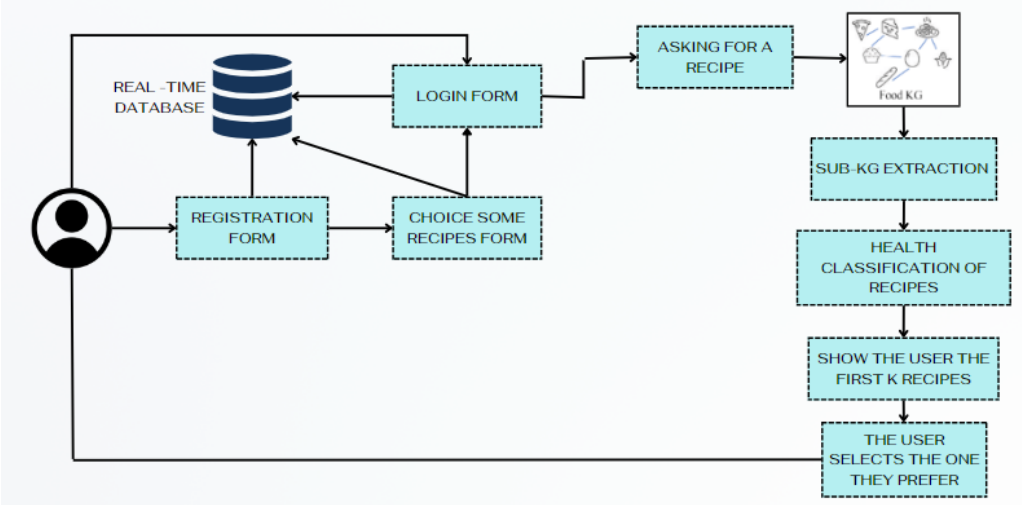

Figure 4.1: The Healthy Food Recommendation System over Semantic-Driven Knowledge Graph Pipeline

Based on this initial user interaction with the system, it is concluded that a Content-based Recommendation System could be developed that would consider both user preferences and the information contained in the FoodKG subgraph.
The proposed solution will be described in more detail in the following paragraphs.

4.2 FoodKG Subgraph Extraction

The first step towards the development of the Recommendation System in question is the extraction of a sub-graph from FoodKG.
FoodKG, in effect, is a vast knowledge graph consisting of 1 029 720 recipe nodes and ingredient nodes with various edge types; given its size, constructing a Recommendation System that would take the entire knowledge graph as input would have led to various memory and computational problems.
Furthermore, since not all nodes of the graph could be needed at the same time and with the same user, it is decided to opt for a reduction of FoodKG concerning food allergies and the diet adopted by the user.
Consequently, taking advantage of the new labels associated with the presence of specific allergens or the unsuitability of the ingredient for a specific diet, a series of SPARQL queries are generated, one for each food allergy and the two diets considered. In particular, an intersection mechanism of the results of these queries is implemented that could take into account the allergy and diet information entered by the user during registration. In this way, it is possible to obtain from FoodKG, a sub-graph containing all recipe nodes consisting only of ingredients suitable for the user, which will be the input of the future Recommendation System.

4.3 The GraphSage Algorithm

Once the graph has been created, significant relationships (edges) between all entities (nodes) in the graph have been incorporated. The next question that comes to mind is to find a way to integrate information about the structure of the graph (e.g. information about the node's global position in the graph or the structure of its local neighbourhoods) into a Machine Learning Model so that the goal associated with the creation of a Recipe Recommendation System can be achieved.
One way to extract structural information from the graph is to calculate graph statistics

using node degrees, clustering coefficients, kernel functions or manually designed features to estimate the structures of local neighbourhoods. However, end-to-end learning cannot be performed with these methods, i.e. features cannot be learned with the help of the loss function during the training process.
Therefore, to address this problem, representation learning approaches are adopted to encode structural information on graphs in Euclidean space (vector/embedding space).
The key idea behind Graph Representation Learning is to learn a mapping function that incorporates nodes, or whole (sub)graphs (from non-Euclidean), as points in low-dimensional vector space (in embedding space). The aim is to optimize this mapping so that nodes that are neighbours in the original network also remain close to each other in the embedding space (vector space), pushing unconnected nodes away. Thus, by doing so, the geometric relationships of the original network within the embedding space can be preserved by learning a mapping function. The diagram in Figure 4.2 illustrates the mapping process; the encoder maps nodes u and v to the low-dimensionality vectors zu and zv:

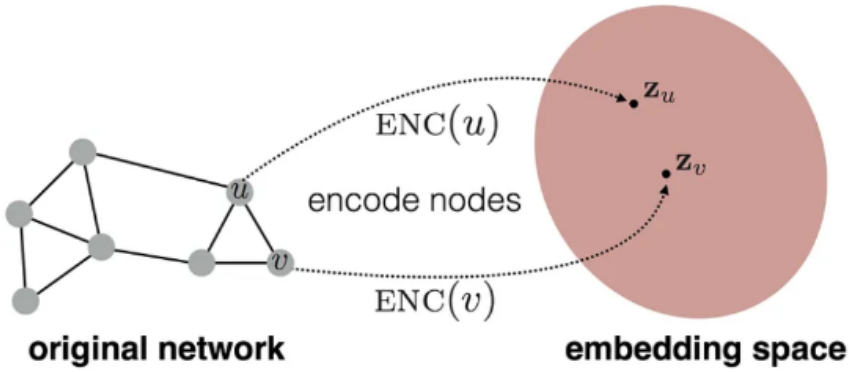

Figure 4.2: Mapping process in learning graphic representation

In particular, the GraphSage (Sample and Aggregate) algorithm, an inductive deep learning method (it can generalize to invisible nodes) developed by Hamilton, Ying and Leskovec in 2017 [22], is chosen to generate dimensional vector representations for graph nodes.
It, indeed, is a Graph Neural Network (GNN) model designed to work with graph-structured data, where nodes represent entities and edges represent relationships or connections between entities. It is particularly useful when dealing with graphs where nodes have associated features and it is desired to learn embeddings or node representations for downstream tasks as it samples neighbouring nodes, aggregates information from them and updates node representations across multiple layers to learn meaningful embeddings.

Furthermore, GraphSage is an inductive algorithm for computing node embeddings that uses information about node characteristics to generate node embeddings on invisible nodes or graphs. Instead of training individual embeddings for each node, the algorithm learns a function that generates embeddings by sampling and aggregating features from a node's local neighbourhood.
However, this algorithm contrasts previous graph Machine Learning methods such as Graph Convolutional Networks or DeepWalk, which are inherently transductive, i.e. they can only generate embeddings for nodes in the fixed graph during training. This implies that if, in the future, the graph evolves and new nodes (not seen during training) make their way into the graph, then it is necessary to retrain the entire graph to compute embeddings for the new node. This limitation makes transductive approaches inefficient to apply on evolving graphs (such as social networks, protein-protein networks, etc.) due to their inability to generalize over invisible nodes. The other main limitation of transductive approaches (mainly DeepWalk or Node2Vec) is that they cannot exploit node characteristics, e.g. text attributes, node profile information, node degrees, ...
On the other hand, the GraphSage algorithm exploits both the rich node features and the topological structure of each node's neighbourhood to efficiently generate representations for new nodes without retraining.
Some of the widespread use cases of GraphSage are:

- Dynamic graphs: graphs that evolve over time such as social network graphs on Facebook, Linkedin or Twitter or posts on Reddit, users and videos on YouTube.
- Node embeddings generated via the unsupervised loss function can be used for various downstream Machine Learning tasks such as node classification, clustering and link prediction.
- Real-world applications requiring the calculation of embeddings for their subgraphs.
- Protein-protein interaction graphs: here, the trained embedding generator can predict node embeddings for data collected on new species/organisms.
- UberEats: harnesses the power of Graph ML to suggest to its users the dishes, restaurants and cuisines they might like next. Uber Eats uses the GraphSage algorithm to make these recommendations due to its inductive nature and ability to scale to billions of nodes.
- Pinterest: uses the power of PinSage (another version of GraphSage) to provide visual recommendations (pins are visual bookmarks, e.g. to buy clothes or other products).

PinSage is a random path-based GraphSage algorithm that learns embeddings for nodes (in the billions) in web-scale graphs.

The working process of GraphSage is mainly divided into two steps, the first is performing **neighbourhood sampling** of an input graph and the second one **learning aggregation functions** at each search depth.
In order to understand the importance of neighbourhood sampling, it is necessary to start from the perspective of the Graph Convolutional Network (GCN) diagram: GCN is an algorithm that can exploit both the topological information of the graph (i.e. node neighbourhoods) and node features and then distil this information to generate representations of nodes or embeddings of dense vectors. The diagram in Figure 4.3 intuitively depicts the working process of GCNs.

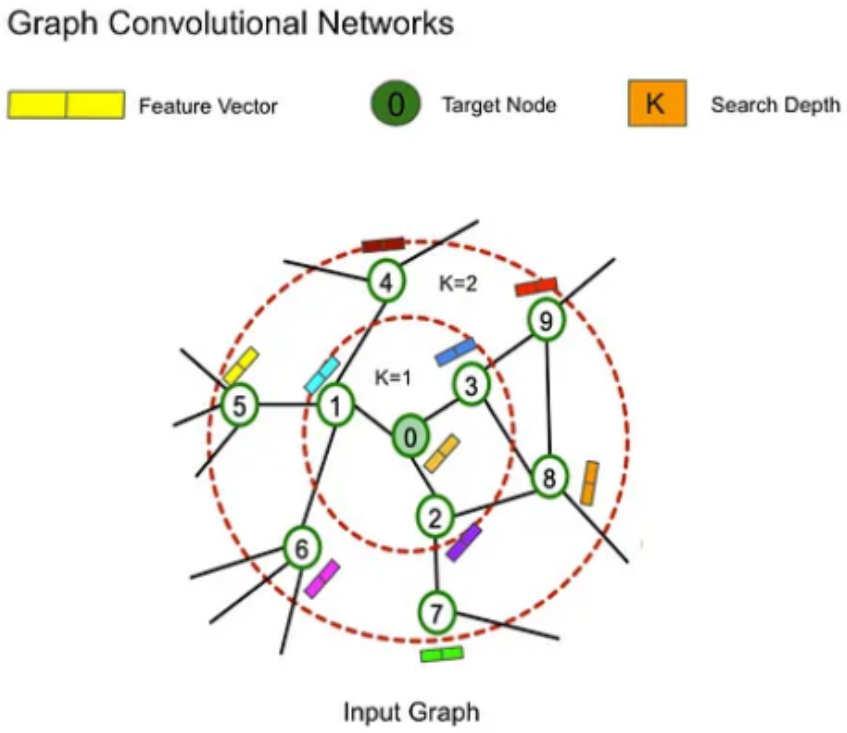

Figure 4.3: The Input Graph

In fact, it starts with an input graph in which nodes are represented by the corresponding feature vectors (e.g. node degree or text embeddings, ...). It is now necessary to

define the search depth (K), which informs the algorithm to what depth it should gather information from the neighbourhood of a target node. Here, K is a hyper-parameter and also represents the number of levels used in GCNs.
At K=0, GCNs initialize all node embeddings on their original feature vector. If one wanted to compute embeddings for target node *0* at level K=1, it would be necessary to aggregate (permutationally invariant function concerning its neighbours) all feature vectors of nodes (including itself) that are at 1-hop distance from node *0* (in this time step or level, the original feature representations of nodes that are at K = 0 are aggregated). For target node *0*, GCNs use an average aggregator to calculate the average of the neighbouring node's features and its own (self-loop) features. After K = 1, target node *0* knows information about its immediate neighbour; this process is shown in Figure 4.4.

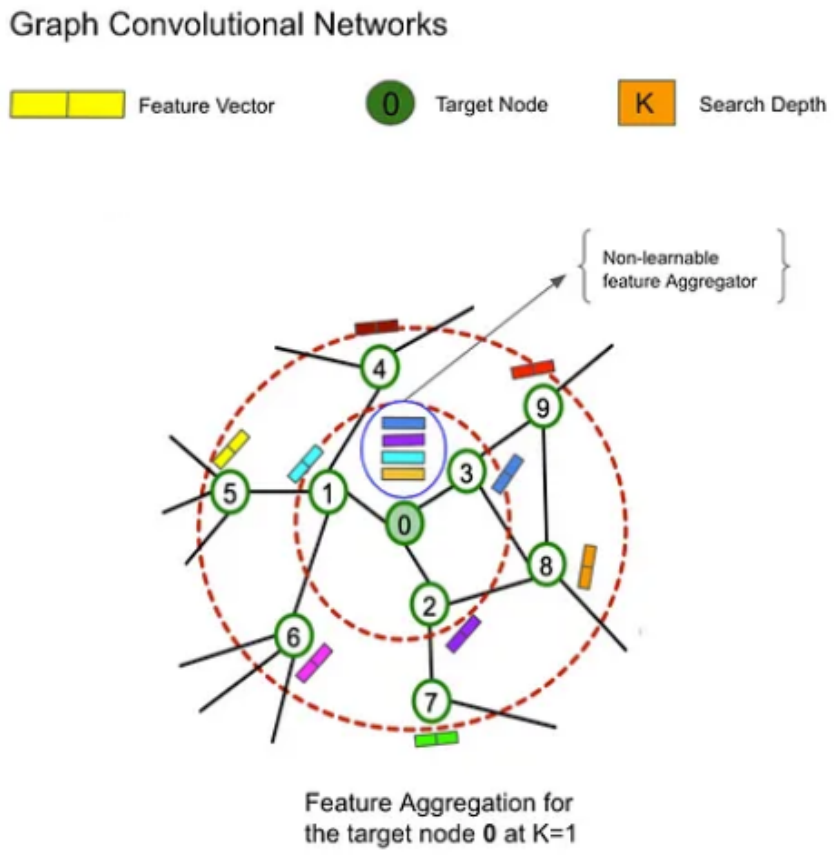

Figure 4.4: The working process of GCNs

This process is repeated for all nodes in the graph to find new representations for each node on each level. In particular, as the search depth increases, the scope of the target node in terms of aggregation of features in its local neighbourhood also increases.

As mentioned earlier, GCNs calculate node representations using neighbourhood aggregation. For training purposes, it is possible to represent the k-hop neighbourhood of a target node as a computational graph and send these computational graphs in mini-batch mode to learn the network weights (i.e. by applying stochastic gradient descent). The Figure 4.5 illustrates a computational graph for the target node from 0 up to the 2-hop neighbourhood.

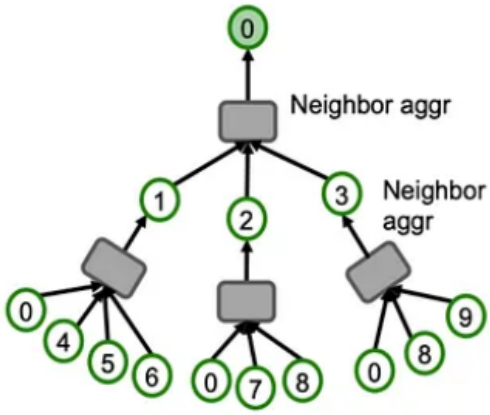

Figure 4.5: Computational graph for the target node from 0 up to the 2-hop neighbourhood

Now, the problem with this is that:

1. *Computationally Expensive:* Since each node has to generate the complete K-hop neighbourhood computational graph and, then, it needs to aggregate plenty of information from its surroundings. As one deepens, the neighbourhood (large K) computation graph becomes exponentially large. This could lead to a problem fitting these big computational graphs inside GPU memory.

2. *The curse of Hub nodes or Celebrity nodes:* Hub nodes are those nodes which are very high degree nodes in the graph, e.g. a trendy celebrity having millions of connections. If that is the case, it is necessary to aggregate the information from millions

of nodes to compute the embeddings for the hub node. Therefore, the generated computational graph for the hub node is very huge. This problem is illustrated in Figure 4.6:

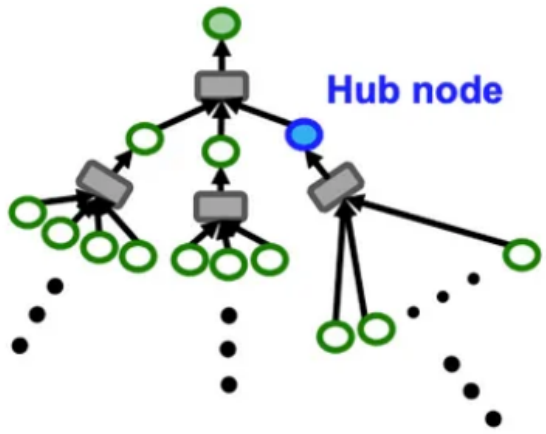

Figure 4.6: The huge generated computational graph for the hub node problem

Therefore, the idea is not to take the entire K-hop neighbourhood of a target node but to select a few random nodes from the K-hop neighbourhood to generate a computational graph. This process is known as neighbourhood sampling, which gives the GraphSage algorithm its unique ability to scale up to a billion nodes in the graph. Therefore, using this approach, if a hub node is encountered, the entire K-hop neighbourhood is not considered, but rather, a few random nodes are selected from each level or depth of K-search. Now, the generated computational graph is more efficient to handle by the GPU. The Figure 4.7 shows this process by sampling at most two neighbours at each hop.

Furthermore, GraphSage is an inductive version of GCNs, which implies that it does not require the entire graph structure during learning and can generalize well to invisible nodes. It is a branch of Graph Neural Networks that learns node representations by sampling and aggregating neighbours from multiple depths or search hops. Its inductive property is based on the premise that one does not need to learn embeddings for each node but instead learns an aggregation function that, when provided with information from a node's local neighbourhood, then knows how to aggregate those features in such that the

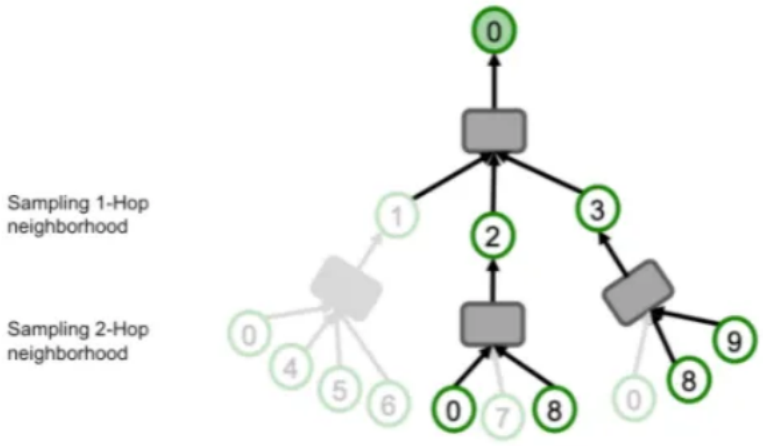

Figure 4.7: The Neighbourhood sampling process

aggregated feature representation of a node v now includes the information about its local surroundings or neighbourhood. It can be said, therefore, that GraphSage is different from GCNs in two ways, namely:

1. Instead of taking the entire K-hop neighbourhood of a target node, GraphSage first samples or deletes the K-hop neighbourhood computation graph and then performs the feature aggregation operation on this sampled graph to generate embeddings for a target node.

2. During the learning process, GraphSage learns the aggregator feature to generate node embeddings while GCNs use the symmetrically normalised Laplacian graph.

The Figure 4.8 illustrates how GraphSage node *0* aggregates information from its sampled local neighbours at search depth K=1. If one looks at the graph on the right, one finds that at K=1, target node *0* now has information about its neighbours down to 1-hop.

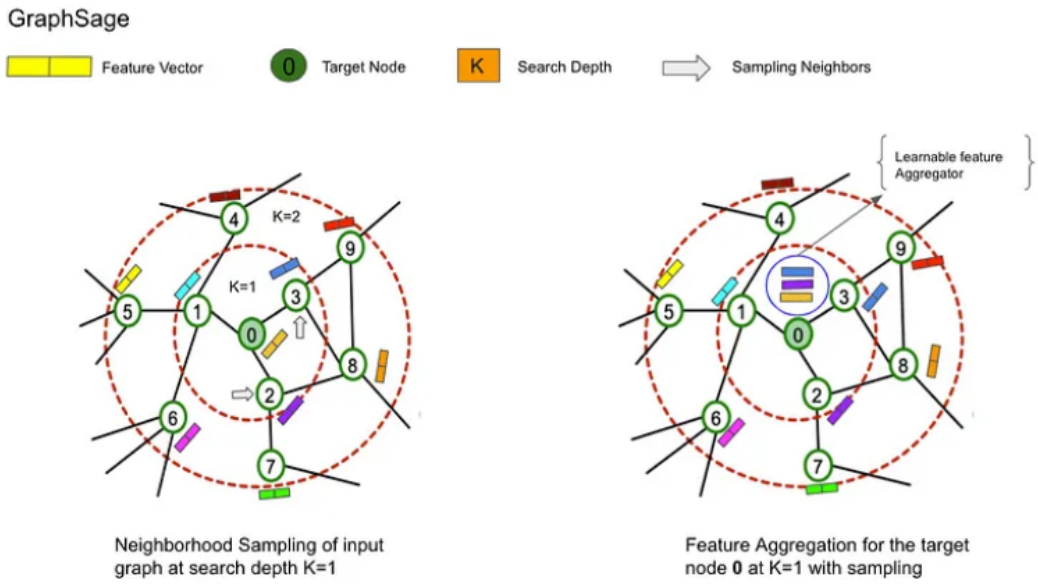

Figure 4.8: GraphSage nodes aggregation

4.3.1 Formal Explanation of GraphSage

The key concept of GraphSage is to learn how to aggregate feature information from a node's local neighbourhood. Now, it can be understand more formally how GraphSage generates node embedding at each layer (K) using forward propagation. It is now necessary to define some notations:

- x_v = Feature vector of a node **v**
- h_v^0 = Initial node embeddings representation for a node **v** (original feature vector e.g. text or image embeddings)
- h_v^k = Node embedding representation for a node **v** at the K-th layer or search depth
- z_v = Final representation of a node **v** after K layers
- V = Set of nodes in the graph
- W^K = Weight matrix at the K-th layer
- $N : v \rightarrow 2^v$ = Neighborhood function

As seen in the GraphSage diagram above at K=1, target node *0* aggregates information (features) from its local neighbours up to 1-hop. Similarly, at k=2, target node *0* aggregates information from its local neighbours up to 2-hops, i.e. it now knows what is in its neighbourhood up to 2-hops. Therefore, one can iterate this process whereby target node *0* incrementally obtains more and more information from further points in the graph. One performs this process of gathering information for each node in the original graph ($\forall$ v $\in$ V). Some visual elements are added to understand this iterative process in a much more intuitive way.
The Figure 4.9 illustrates the calculation graph of a target node *0* at level K=0, at which point all nodes in the graph are initialized on their original feature vectors. The goal is to find the final representation of node *0* (i.e. z_0) at level K=2 through an iterative local neighbourhood information-gathering process. This iterative process is sometimes also known as the **message-passing approach**.

$$h_v^0 \leftarrow x_v, \forall v \in V$$

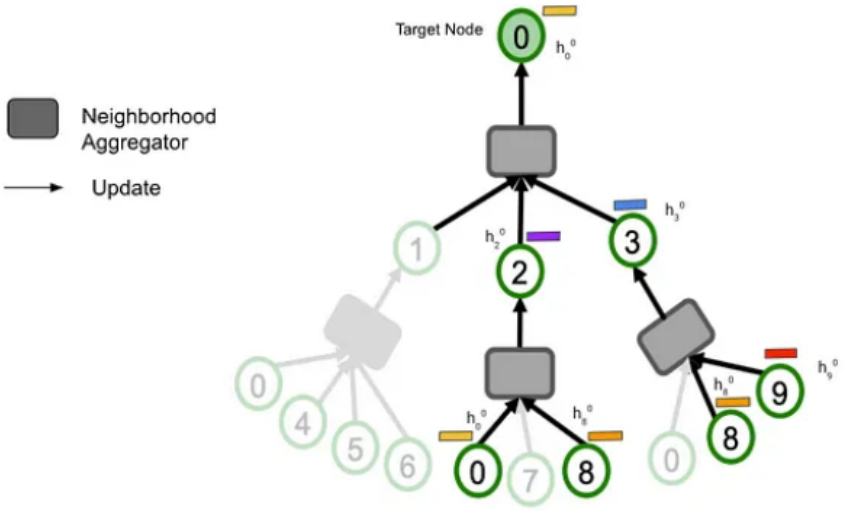

Figure 4.9: Computation Graph at K=0

Neighbourhood aggregation at K=1

Since nodes incrementally gather information from the deepest depths of the graph, the iteration process from the search depth 1....K. At K=1, the representations of neighbouring nodes for target node *0* (h_0^1) are aggregated, i.e. the representations of nodes *2* and *3* that are at the previous layer (h_2^{K-1} and h_3^{K-1}) into a single vector. Here (h_0^1) is an aggregated representation. In the same time step, nodes *2*, *3* and *9* will also aggregate the feature vectors from their local neighbourhoods to a distance of 1-hop. Now, at this point, each node in a computation graph knows what kind of information is in its immediate neighbourhood. Therefore a formal representation of this step can be given as:

$$h_{N(v)}^k \leftarrow AGGREGATE_k(\{h_u^{k-1}, \forall u \in N(v)\})$$

Updation

Once the aggregate representation, i.e. h_0^1, has been obtained, the next step would be to concatenate or combine this aggregate representation with its representation from the previous level (h_0^0). Then, the transformation is applied to this concatenated output by multiplying it with a weight matrix **WK**; it is possible to think of this process as similar to applying convolutional kernels (learnable weight matrices) on images to extract features. Eventually, a non-linear activation function is applied to this transformed output, making it capable of learning and performing more complex tasks. Specifically, the GraphSage algorithm learns the weight matrix individually at each search depth K or one could also say that it learns how to aggregate information from the vicinity of a node at each search depth. It is, therefore, possible to formally represent this step as:

$$h_v^k \leftarrow \sigma(W^k \cdot CONCAT(h_v^{k-1}, h_{N(v)}^k))$$

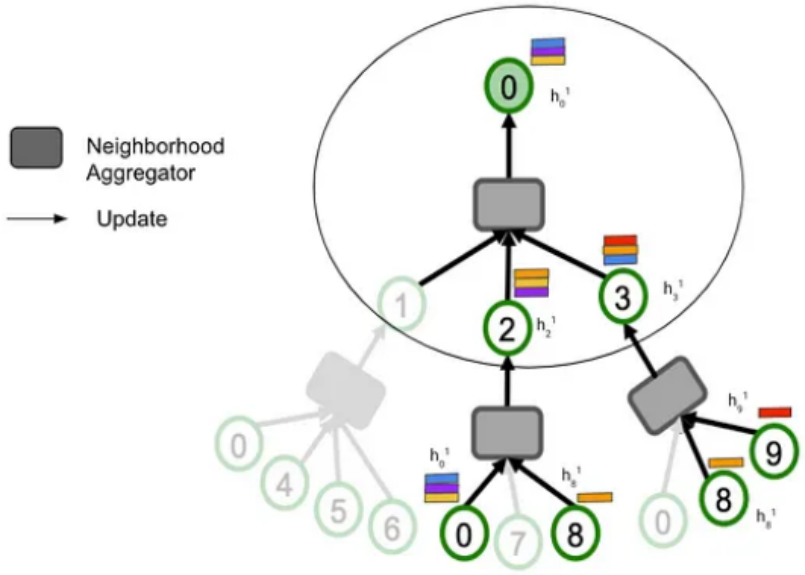

Figure 4.10: Computation Graph at K=1

Normalizing Node Embeddings

Next, normalization is applied to the node representation h_v^K (or, at this time, step h_0^1), which helps the algorithm maintain the general distribution of node embeddings. This step is calculated as:

$$h_v^k \leftarrow \frac{h_v^k}{||h_v^k||_2}, \forall v \in V$$

Node Embeddings at K=2

The information collection from the node's local neighbourhood at K=1 is completed. At K=2, the nodes explore the farthest points of the graph, i.e. going beyond their immediate surroundings and examining a jump distance of 2. Once again, the aggregation of the node's local neighbourhoods is performed, but this time, the target node *0* will now have the information of its neighbours located at distances of 1 and 2 jumps. Then, the update and normalization process is repeated for search depth K=2. Since K=2 is set to understand the flow of the GraphSage algorithm, the algorithm stops here. After K=2, each node in the calculation graph is represented by the respective embeddings of the final nodes, i.e. *zv*. This workflow is displayed below in Figura 4.11:

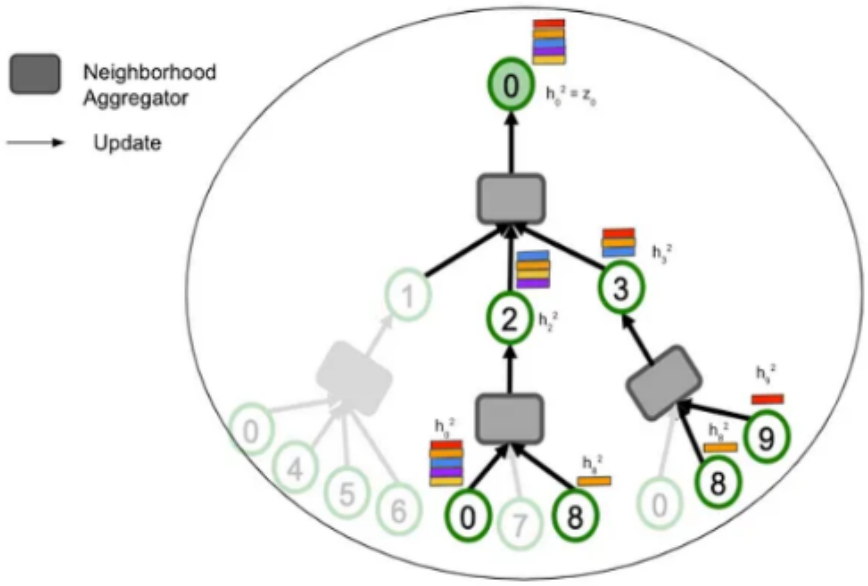

Figure 4.11: Computation Graph at K=2

The Figure 4.12 shows the process associated with the GraphSage algorithm just described.

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Figure 4.12: GraphSage Algorithm

The following paragraphs will describe in more detail the necessary steps that led to the implementation of the GraphSage model.

4.4 Data Pre-Processing

First of all, it is necessary to organize the data in a suitable format. More precisely, given the considerable size of FoodKG, the main nodes and edges are selected to be considered during the learning of GraphSage, in particular the title of the recipes and their ingredients as they are fundamental in the description of a recipe and also the healthiness of the recipe so that the algorithm can also consider the health aspect during its learning. In this regard, converting the selected data into feature vectors is necessary. Concerning the textual character data, they initially underwent a process of converting all the characters from which they are composed into lowercase. Subsequently, using the Nltk library's *word_tokenize*, it is possible to tokenize the textual data, breaking them down into smaller, more meaningful units, the tokens based on spaces or whitespace characters.
This is followed by the removal of punctuation and stopwords, i.e. the most common words, to be able to reduce the dimensionality of the data; and, before converting the data into feature vectors, lemmatization is also applied, which aims to convert words into their basic form, taking into account the part of speech (POS) of a word.

This is a crucial step in NLP as it transforms raw text data into a format suitable for further analysis and modelling, enabling machines to understand and process human language effectively.
Now, using the following techniques, it is possible to convert the selected data into feature vectors:

- For the recipe titles, the TF-IDF algorithm is applied using the *TfidfVectorizer* function of Sklearn.feature_extraction, as it attaches greater importance to terms that are less frequent in the document corpus but more relevant to the search [12].
  The algorithm's name is composed of two terms: TF and IDF. TF stands for Term Frequency, while IDF stands for Inverse Document Frequency. The method was devised in 1972 by S. Jones, who defined an initial IDF formula. Subsequently, the algorithm was refined in 1975 by Salton with the addition of the TF component. The algorithm: Given N documents in a database, one decides on a term K to be searched. According to Jones, the effectiveness of the extrapolation process is inversely proportional to the homogeneous distribution of the term in the documents. If the term $K$is present in all N documents in the database, the effectiveness of the search process is low, as the word K is common in all resources (e.g. articles, prepositions, ...). Conversely, if the word K is contained in only a few documents, the effectiveness of the selection becomes higher. The search process returns the few pages that contain the term.
  The calculation of IDF (Inverse Document Frequency): This part of the algorithm concerns the IDF component, which can be summarized with the following formula:

$$IDF = \frac{logN}{n_k}$$

  The calculation of TF (Term Frequency): The value of TF is determined by the ratio between the number of occurrences $O_{K,X}$ of the term K in document X and the number D_X of the words in document X (document size).

$$TF = \frac{O_{K,X}}{D_X}$$

  Calculating TF-IDF: To calculate TF-IDF, simply multiply TF and IDF together as in the following:

$$TF - IDF = TF_{K,X} - IDFK$$

  The TF-IDF parameter is high when the word is commonplace in a document, and

the term is not present in all the documents in the database. This makes it possible to reduce the importance of common words (e.g. of, the, a, ...) and enhance the other terms in the user query.

- For the ingredients that make up a recipe, on the other hand, a popular Word2Vec Natural Language Processing (NLP) word embedding technique developed by Google researchers is applied. [13]
  In fact, using the Word2Vec function of the Gensim library, it is possible to obtain word embeddings, i.e. dense vector representations of words in a continuous vector space in which semantically similar words are close to each other. In particular, Word2Vec is extremely powerful in capturing the semantic relationships between words in large text corpora based on their context.

- Using the function of the sklearn.preprocessing library [7], it attempts to rescale the healthiness values of recipes from the range $[4, 12]$ to the range $[0, 1]$. The transformation is given by applying the following functions:

$$Xstd = \frac{(X - X.min(axis = 0))}{(X.max(axis = 0) - X.min(axis = 0))}$$

$$Xscaled = Xstd * (max - min) + min$$

At this point, since FoodKG is characterized by the fact that there is no direct edge between recipe nodes, then, from the feature vectors just generated, it is conceivable to create potential edges between recipes as well. More precisely, similarity is calculated between recipe titles and also between the ingredients that make up each recipe.
Among the different ways of calculating the similarity between two feature vectors, cosine similarity is used, a similarity metric commonly used in Natural Language Processing (NLP) and Information Retrieval that has the characteristic of being scale-invariant, i.e. it is not affected by the size of the vectors being compared and only considers the direction of the vectors [3].
As can be seen from Figure 4.13, it measures the cosine of the angle between two vectors in a high-dimensional space: if two vectors point in the same direction (i.e. are similar), the cosine of the angle between them is close to 1. If they are orthogonal (i.e. dissimilar), the cosine similarity is close to 0. The measure aligns well with the notion of similarity in NLP, where words or documents are considered similar if they share similar directions

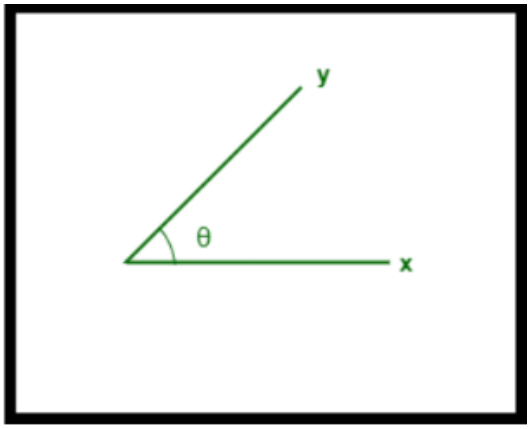

Figure 4.13: The cosine similarity and the angle between two vectors

in vector space. Furthermore, cosine similarity calculation is computationally efficient, especially when working with large text datasets and involves simple scalar products and vector quantities, making it faster to calculate than other similarity measures.

The formula for finding the cosine similarity between two vectors is:

$$(x, y) = \frac{x \cdot y}{||x| \times ||y||}$$

Where:

- $x \cdot y$ = product (dot) of the vectors x and y.
- $||x||$ *and* $||y||$ = length (magnitude) of the two vectors x and y.
- $||x|| \times ||y||$ = regular product of the two vectors x and y.

The cosine similarity between two vectors is measured in θ.
Once the cosine similarity on the recipe titles and recipe ingredients had been calculated, the results could be combined using the following formula:

$$alpha * title\_sim + (1 - alpha) * ingredient\_sim$$

in which both vectors are weighted equally (alpha = 0.5). From the results obtained, an edge is inserted between two recipes in the graph if the similarity value between the two is more significant than a threshold set at 0.7. In addition, in order to incorporate the health

aspect between two similar recipes, a weight is attached to the newly inserted edges, which considers the healthier recipe between the two to be of greater relevance: Given two recipes, $recipe_1$ and $recipe_2$ and the associated health values $health\_value_1$ and $health\_value_2$, the difference between $health\_value_2$ and $health\_value_1$ is first calculated:

$$health\_dif = health\_value_2 - health\_value_1$$

Then, a scaling factor is defined to adjust the edge weights:

$$scaling\_factor = \frac{1.0}{(12 - 4)}$$

and finally calculate the weight to be assigned to the created edge:

$$weight\_adjustment = 1.0 - (health\_dif * scaling\_factor)$$

4.5 Neighbour Sampling

Neighbour sampling is a critical component for the implementation of a GraphSage algorithm that is used for node classification and learning of representations in structured graph data. It is performed to efficiently generate a set of neighbour nodes for each target node in the graph, allowing the algorithm to learn node representations by aggregating information from these neighbours.
At this stage, a fixed number of neighbours are randomly sampled for each node in the graph by adopting a sampling strategy to explore the local neighbourhood of each node. Typically, the most commonly used sampling strategy is that of random walks, sequences of nodes in the graph in which each step is probabilistically chosen based on the neighbours of the current node.
At each step of the sampling process, indeeed, a fixed number of neighbouring nodes are selected to be included in the sampled neighbourhood; these neighbours can be selected uniformly at random or according to specific criteria, such as node degrees or edge weights. A basic overview of a random walk algorithm is given by:

- *Initialization:* One starts from an initial state or node of a system. This initial state can be a position in a graph, a point in a space or any other relevant starting point.
- *Transition probabilities:* One defines transition probabilities or rules that determine the next state or node based on the current state. These probabilities describe the probability of moving from the current state to any possible neighbouring state.

- *Random selection:* At each step of the walk, the next state or node is randomly chosen based on the defined transition probabilities. This step introduces randomness into the process.
- *Repetition:* The process continues for a specific number of steps or until a stopping criterion is met. The sequence of states visited during the walk forms a random path.

It is a common technique in Graph Neural Networks (GNNs) and graph-based Machine Learning to explore the local neighbourhood of each node in a graph in order to acquire important information about the structure and connectivity of a node's neighbours. Through this technique, it is possible to aggregate information from nodes that are directly or indirectly connected to the target node by learning meaningful representations of the nodes. Moreover, it also reduces the computational load by allowing the possibility to focus the learning only on a subset of neighbours instead of considering the entire graph. At the same time, thanks to the element of randomness, it improves the robustness of the model to be developed.

In this case, the parameters for setting the hyper-parameters and sampling the neighbours are defined before arriving at the implementation of the random walk algorithm. In fact, concerning the length of the random walks and the number of random walks, a list is defined containing different values, respectively 5, 10, 15 and 20 for the former and 5, 10, 20 and 30 for the latter. On the other hand, the number of neighbours to be selected for each node is set directly to 10. At this point, the hyper-parameter setting cycle is used to iterate different combinations of these values in order to be able to identify the optimal values of these hyper-parameters. In particular, after evaluating all the combinations, the optimal value for the length of the random walks is 5, and for the number of random walks, 30. These optimal parameters are then used to generate the sampled neighbours for each node, which will, later, form part of the input for training the GraphSage model.

4.6 Data Augmentation

Before proceeding with the definition and training of the GraphSage model, considering the small number of user interactions with the FoodKG subgraph, the possibility of adopting data augmentation techniques to increase the size of the dataset artificially has been evaluated. In fact, only one user's data is available, who, as already mentioned, expressed a preference for five of the proposed recipes during the registration phase. Given the small

number of user interactions, it is decided to apply these techniques to the entire labelled dataset and not only the training set so that an overall dataset is a little larger and both training and model evaluation can benefit.
Then, one proceeds by creating variations of the existing labelled data: in this case, the technique of text augmentation based on replacing certain words in recipe titles or recipe ingredients with their synonyms is used to diversify the textual content of the recipes. More precisely, *WordNet*, a lexical database that organizes words into synonyms (sets of synonymous words), has been used to generate the synonyms, and thanks to its combination with the Nltk library, it has been possible to access the synonyms associated with some of the words that make up the titles or ingredients of the recipes. At this point, in a similar manner to that described above, the relevant feature vectors have been generated for this new data and joined to the same set of neighbouring nodes as the original recipe.

4.7 GraphSage Architecture

Before proceeding with the description of the GraphSage model architecture for developing the Recipe Recommendation System, it is necessary to focus momentarily on user interactions. Although data augmentation techniques have made it possible to expand the dataset, the number of user interactions is still relatively small. If one wanted to convert the recommendation problem into a binary classification problem in which one wanted to go and predict which recipes the user might or might not like, the conventional supervised learning paradigm, which relies heavily on labelled data for training, would be severely hampered by the scarcity of labelled interactions. Consequently, in response to this limitation, semi-supervised learning techniques are chosen.
Indeed, semi-supervised learning allows for the effective exploitation of the unlabelled dataset. Underlying this is the concept of self-training, a powerful technique that iteratively labels unlabelled data points using model predictions. In this way, the GraphSage model will progressively improve its understanding of the underlying patterns and nuances in the data.
To this end, two distinct datasets are created: the first containing labelled data, namely data for which there is an interaction with the user for which the user has been able to express a preference or otherwise for a given recipe; the second containing unlabelled data as it has not yet been submitted to the user. Therefore, the idea is to develop a binary classification model (preferences - 1, non-preferences - 0) that can predict which recipes might satisfy the user's preferences.
In the following, an in-depth exploration of the architecture of the proposed GraphSage

model is provided, clarifying the main components.

The GraphSage model is a versatile graph-based representation learning framework suited to capturing rich structural and semantic information within complex graphs. Indeed, the core of the Recommendation System is encapsulated in the GraphSage class, a neural network model meticulously tailored to address the complexities of the Recommendation problem. The initialization of this class serves as the basis on which the model architecture is built. The GraphSage class is anchored in the TensorFlow ecosystem, exploiting the capabilities of the *tf.keras.Model* module. It is worth noting that this class is not simply a stand-alone Python class but, instead, a specialized neural network model that inherits from *tf.keras.Model*. This inheritance is crucial for several reasons:

- *Using the Keras API:* TensorFlow's Keras API is renowned for its intuitive approach to modularizing the construction of neural networks. Inheriting from *tf.keras.Model*, the GraphSAGEclass integrates perfectly with Keras, offering the advantages of a high-level neural network API.
- *Customization:* Inheriting from tf.keras.Model provides the flexibility to customize the behaviour and architecture of the neural network. Model layers, training procedures and forward passages can be defined while respecting the design principles of Keras.
- *Interoperability:* The GraphSage model interfaces seamlessly with the broader TensorFlow ecosystem, facilitating compatibility with various TensorFlow tools, such as TensorBoard for visualization, TensorFlow Serving for distribution, and TensorFlow's rich collection of pre-processing and data augmentation utilities.

The initialization of the GraphSAGEclass also acts as a gateway to define the hyper-parameters that govern the behaviour of the model:

- *Num_neighbors:* This hyper-parameter defines the number of neighbours to consider when aggregating graph information. It affects the extent to which the model captures local contextual information.
- *Title_feature_dim, Ingr_feature_dim, Health_feature_dim:* These parameters specify the dimensions of the input features associated with titles, ingredients and health attributes. They model the input layers of the model.

- *Hidden_dim:* The dimensionality of the hidden layers in the model, which plays a crucial role in capturing complex patterns within the data.
- *Num_classes:* This hyper-parameter defines the number of output classes in cases where the suggestion problem involves several classes or categories. In this case, this is a binary classification task; consequently, this value is set to 1.
- *Dropout_rate:* Dropout is a regularization technique that prevents overfitting. It determines the dropout rate, the probability of neurons dropping out during training.
- *Activation:* The choice of activation function, which introduces non-linearity into the model.

In summary, the initialization of the GraphSAGEclass indicates its integration within the TensorFlow ecosystem through inheritance from *tf.keras.Model*. It also lays the foundation for defining essential hyper-parameters that configure the model's behaviour and ultimately contribute to its effectiveness in addressing the recommendation problem.
At this point, the input layers are defined: these layers define the data types or functionalities on which the model operates, including the functionalities of the target nodes and their neighbours. There are three different input layers, one for each feature representing a recipe: input layers for recipe titles and their neighbours, input layers for ingredient features and their neighbours, and input layers for health features and their neighbours. These input layers provide feature vectors for nodes and their neighbours in the graph.
This configuration allows the incorporation of information from neighbours during aggregation. Indeed, the aggregation layer constitutes a crucial point in the GraphSage model and is responsible for synthesising information from neighbouring nodes, a process that is essential for capturing the essence of a node's local context. In this case, an average pooling layer is used that aggregates the features of neighbouring nodes and averages per element of the neighbouring feature vectors to obtain a summary representation for each node.
Subsequently, three dense hidden layers are defined that apply various transformations to the aggregated features to create informative and discriminating representations.
Each hidden layer is followed by batch normalization and an activation function specified by the hyper-parameter: the former helps stabilize and accelerate the training of deep neural networks by normalizing the input on one layer for each mini-batch during training. It also helps mitigate the evanescent gradient problem and allows for faster convergence while reducing the risk of overfitting; the second, on the other hand, allows for the introduction of non-linearity into the model, enabling it to capture complex relationships within the data.

After each hidden layer, dropout regularization is applied to prevent overfitting of the model. This is, in fact, a regularization technique that helps prevent overfitting by randomly dropping out a fraction of neurons in a layer during each forward and backward step. This dropping of neurons introduces a form of pattern averaging and reduces dependence on specific neurons. It can be applied to hidden layers to encourage the network to learn more robust and generalized features. Finally, the output layer produces the final predictions using sigmoid as the activation function.
Through the call method, the forward pass of the model is defined. It takes as input the labelled features, the labelled neighbour features and the type of features for both labelled and neighbour nodes and performs the following steps:

- Aggregates neighbouring functionalities using the aggregation level;
- Concatenates labelled and aggregated functionalities;
- Passes concatenated features through hidden layers with batch normalization and dropout;
- Produces the final predictions through the output layer.

At this point, the GraphSage model is built using a HyperParameters object from Keras Tuner to search for the best combination of hyper-parameters:

- Num_neighbors indicates the number of samples to be considered in the construction of the model. It accepts a minimum of 2 neighbours up to a maximum equal to the total number of neighbours;
- Hidden_dim represents the dimensionality of the hidden layers within the model by determining the number of neurons or units in each hidden layer of the model. The model could range from a minimum of 64 to a maximum of 512 neurons.
- Dropout_rate controls the dropout regularization applied to the hidden layers of the model. It specifies a range of possible dropout rates, with a minimum value of 0.3 and a maximum value of 0.5.
- Learning_rate determines the step size at which the model weights are updated during the training process. It is a critical hyper-parameter because it influences the convergence and performance of the model during training. The learning rates to choose from are 0.01, 0.001 and 0.0001, respectively.

- Activation specifies the activation function used in the hidden layers of the model. Activation functions introduce non-linearity into neural networks, enabling them to model complex relationships in the data. Different activation functions have different properties, and the choice of activation function can affect model performance and training dynamics. The following options are provided for the Keras Tuner:
  - *ReLU (Rectified Linear Unit):* ReLU is a widely used activation function that is computationally efficient and helps mitigate the escape gradient problem. It is defined as
    $$f(x) = max(0, x)$$
    which means it outputs zero for negative inputs and the input value for positive inputs.
  - *Leaky ReLU:* Leaky ReLU is a variant of ReLU that allows a slight non-zero gradient for negative inputs to prevent some dead neuron problems. It is defined as $f(x) = x$, for $x > 0$ and $f(x) = 0.01 * x$, for $x <= 0$, where 0.01 is a small positive constant.
  - *Tanh (hyperbolic tangent):* The activation function tanh compresses input values in the interval $[-1, 1]$. It is often used in scenarios where outputs must fall within a limited range.
- Optimizer determines the optimization algorithm used to update the model weights during training. The choice of optimizer is crucial because it affects the speed of convergence and the quality of the trained model. There are three optimizers from which the model can choose Adam, Rmsprop and SGD, which represent three different optimization algorithms:
  - *Adam (Adaptive Moment Estimation):* Adam is a popular optimizer that combines the advantages of two other optimization techniques, RMSprop and Momentum. It adapts the learning rates for each parameter during training and often converges quickly.
  - *RMSprop (Root Mean Square Propagation):* RMSprop is an adaptive learning rate optimization algorithm that adjusts learning rates for each parameter based on historical gradient information. It is effective in handling sparse data and mitigates some of the problems of the original gradient descent.
  - *SGD (Stochastic Gradient Descent):* SGD is the classic optimization algorithm that updates the model weights using the gradient of the loss function concerning each parameter. Although it is simple, it can be effective when adjusted correctly but may require careful tuning of the learning rate and other hyper-parameters.

- Loss quantifies the difference between predicted and actual values. It measures how closely the model's predictions align with actual results. The goal during training is to minimize this loss, i.e. to make the predicted values as close as possible to the actual targets. The following options are provided for the Keras Tuner:
  - *Binary crossentropy loss (binary_crossentropy):* This loss function is commonly used for binary classification problems where the model predicts 0 or 1. It measures the dissimilarity between the true binary labels and the predicted probabilities.
  - *Hinge loss (hinge):* Hinge loss is often used for binary and multiclass classification problems. It is suitable for models performing classification tasks and helps maximize the margin between classes.

Therefore, a GraphSage model is created using the specified hyper-parameters, which is then compiled with the other found hyper-parameters.

4.8 GraphSage Training

Before training the newly created GraphSage model, it is necessary to split the labelled dataset into training and test sets by performing stratified sampling.
This sampling method is often used in situations where one wants to ensure that the sample represents the diversity of the population, particularly when one has unbalanced data or wants to maintain the same proportion of specific categories or groups in the sample as in the overall population. Therefore, data from each subgroup are randomly sampled, with the same proportion of each subgroup in the sample as in the population. This ensures that the sample is representative of the entire population.

At this point, to address the problem of semi-supervised learning, a function called *self_training* is introduced that performs the self-training of the created GraphSage model. Self-training, indeed, is a simple but effective technique that allows the model to make predictions on unlabelled data and then incorporate these predictions as if they were truth labels. This effectively expands the labelled dataset.
Consequently, it is possible to outline a self-training strategy to improve the performance of a GraphSage model in a semi-supervised learning context by exploiting both labelled and unlabelled data, iteratively refining the model's predictions and learning from the unlabelled data to provide better recommendations.

Specifically, in this case, the self-learning mechanism begins with the initial training of the GraphSage model using only the labelled data available to create an initial model.
Subsequently, the initial model has been used to generate user recommendations that can be considered pseudo-labelled data because they are based on the model's predictions. In this way, it is possible to increase user preferences by combining the limited user interaction data with the pseudo-labelled data generated by the model. At this point, it is necessary to retrain the recommendation model using the expanded dataset.
This process has been repeated iteratively, and at each iteration, the updated model is used to generate new recommendations, treat them as pseudo-labelled data and add them to the training dataset. Expressly, during this process, the number of training epochs is set to 30, and to avoid overfitting and improve the efficiency of the training process, the Early Stopping callback technique is also used. In addition, to avoid the propagation of errors between iterations, forgetting and weight reduction mechanisms are also incorporated for older pseudo-labelled data. To assess the training progress of the model, the model's performance on the test dataset is periodically evaluated.

In summary, the GraphSage model embarks on a journey of continuous improvement, iteratively classifying unlabelled interactions and refining its understanding of user preferences and recipe characteristics. This iterative learning process leads to increasingly accurate recommendations aligned with user expectations.

4.9 GraphSage Performance

After the training process of the GraphSage model, it is possible to test the model on the test set in order to evaluate its performance.
First of all, a representation of the variation in the values assumed by the Accuracy, Precision, Recall and F1-score metrics during the 30 training and validation epochs is displayed in Figure 4.14.

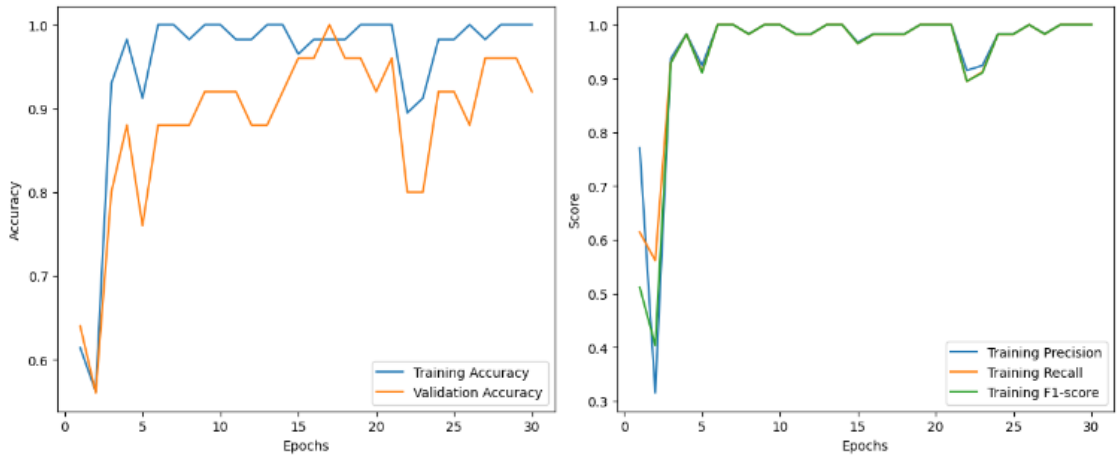

Figure 4.14: Accuracy, Precision, Recall and F1-score training and validation plot

Furthermore, through the use of the evaluate method, it has been possible to evaluate the model's performance with the best combination of hyper-parameters on the test dataset. The Loss and Accuracy values on this data are shown below in Table 4.1.

Test Loss	Test Accuracy
0.8168	0.9200

Table 4.1: Model evaluation results

The loss value represents how well the model's predictions match the actual labels in the test set. A lower Loss value indicates a better alignment between predictions and actual results. In this case, the test Loss of 0.81 suggests that, on average, the model predictions deviate from the actual labels by this value.
The Accuracy metric, on the other hand, indicates the proportion of correctly classified samples in the test dataset. An Accuracy of 92% means that the model correctly predicted the labels for approximately 92% of the test samples.
Indeed, this model performs well even though it may prefer even lower values of the loss value and, at the same time, a higher value of accuracy but, in this case, one has to take into account the problems associated with having a tiny number of labelled data, partly solved by the use of the data augmentation technique and self-training, which also led to having a meagre number of samples in the test set.
Furthermore, although Accuracy provides a clear overview of the model's performance, it is essential to evaluate other metrics such as Precision, Recall and F1 score to understand

better the model's performance, which is why the Table 4.2 shows the result of the classification report generation. It includes Precision, Recall, F1 score and Support metrics for each of the two classes and helps understand model performance on specific classes. By examining Precision and Recall, it is possible to understand the model's strengths and weaknesses in differentiating between classes.

	Precision	Recall	F1-score	Support
0	1.00	0.82	0.90	11
1	0.88	1.00	0.93	14
accuracy			0.92	25
macro avg	0.94	0.91	0.92	25
weighted avg	0.93	0.92	0.92	25

Table 4.2: Classification Report

In particular, it can be seen from the classification report that, as mentioned earlier, the model's overall performance is quite good as it correctly classifies about 92% of the samples. There is also a good value of Precision, the proportion of true and positive predictions out of all positive predictions made by the model, and also of Recall, the proportion of true and positive predictions out of all actually positive instances of the dataset. This suggests that the model effectively makes accurate predictions and identifies actually positive instances.

Furthermore, to get a complete overview of the performance of the newly trained model, the Precision-Recall curve and the AUC curve are also generated. The former is a graphical representation of the trade-off between Precision and Recall for different threshold values used to classify data points. This also allowed the calculation of the Average Precision (AP), a single numerical value that summarizes the area under the Precision-Recall curve and provides a concise measure of the overall performance of a binary classification model across different decision thresholds. A higher AP indicates better model performance. In this case, as reported in Figure 4.15, the AP score is equal to 0.42 which suggests that the system's Precision-Recall trade-off is moderate. It means that the system is reasonably good at retrieving relevant items or detecting objects, but still there is room for improvement.

The ROC (Receiver Operating Characteristic) curve, on the other hand, is a graphical representation of the performance of a binary classification model at various threshold settings. It plots the true positive rate (TPR) on the y-axis and the false positive rate (FPR) on the x-axis for various threshold values. The ROC curve helps visualize the model's ability to distinguish between positive and negative classes. To quantify the model's ability to

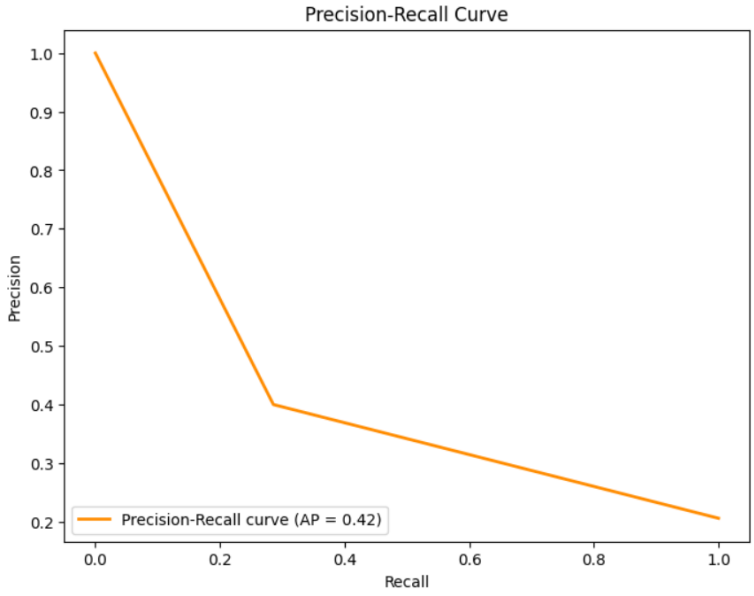

Figure 4.15: Precision-Recall Curve

distinguish between the two classes, the area under the curve (AUC) is used: The AUC is a single numerical value summarizing the model's performance across different threshold settings. A higher AUC indicates a better performance of the model, with a perfect classifier having an AUC of 1. As can be seen from the graph in Figure 4.16, in this case, the AUC value is, which suggests that the model has some discriminative capacity, but can be improved so that it is closer to 1.

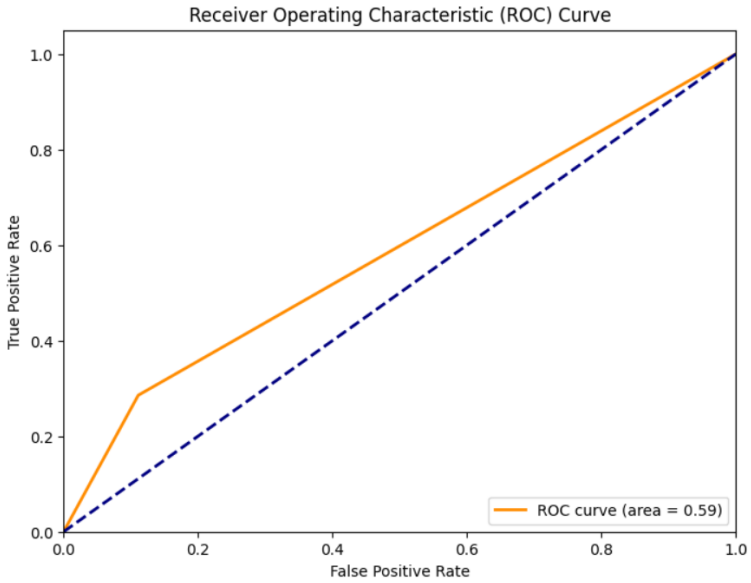

Figure 4.16: Roc Curve

Overall, it has been obtained as a model with a relatively good performance. However, to evaluate it, it is necessary to take into account the starting problems encountered: firstly, having a meagre number of user-recipient interactions and, secondly, the data augmentation and self-training techniques that are applied to solve this problem. Moreover, as time goes by and the number of user interactions collected increases, the Recommendation System generated may increase its performance.

4.10 Calculating Recommendations

After training the GraphSage model using self-learning, it has been possible to generate the content-based recipe Recommendation System using the learned recipe representations.
First, using the trained GraphSage model, feature representations are extracted for all recipes in the FoodKG subgraph. These embeddings represent the recipes in the FoodKG subgraph in a lower dimensional space, capturing their content and relationships.
From the embeddings obtained, it is now possible to extrapolate those related to the user preferences, and, subsequently, the average of the embeddings of the recipes in the user preference list is calculated in order to condense the user preferences into a single vector in the same feature space as the recipe embeddings.

Then, for each recipe, the cosine similarity between the user preference embedding and the recipe embedding is calculated and, based on the results obtained, the recipes are sorted in descending order according to the similarity scores. Consequently, recipes with higher similarity scores (i.e. more similar to user preferences) would appear first.
Finally, based on these similarity scores, the six best recipes are recommended to the user, excluding recipes that are already in the user's preference list to avoid recommending the same recipes that the user has already preferred.

Chapter 5

The Healthy Food KG-based Recommendation System Web App

This chapter describes how the user-friendly user interface is developed and implemented in which users can interact with the generated Recipe Recommendation System. Initially, it will start with a description of the framework that is used for the development of this interface and the introduction of a back-end service that enabled the storage of data acquired from user interactions; it will, then, come to the description of the Web App created with all the associated pages.

5.1 The choice of framework: Streamlit

Web Apps can be created with various programming languages and frameworks. The latter dramatically simplifies the development of an application, but the realization of specific components sometimes requires advanced knowledge and the integration of other programming languages.
Today, the creation of dashboards to visualize and navigate data is increasingly in demand. Unfortunately, this task is on the back burner for data scientists, whose work is mainly oriented towards data analysis. In many cases, people opt for ready-made software that includes the creation of dashboards simply by drag-and-drop. An example of this is *Kibana*, even if it involves using specialized databases, such as Elasticsearch, or formatting the data so that it can be read correctly by the software.
In recent years, *Streamlit*, a free, open-source framework entirely in Python, has become

popular [11]. It allows one to quickly build interactive dashboards and Machine Learning web applications without requiring any front-end web development experience.
Indeed, it is designed to simplify the creation of web applications by data scientists and developers, quickly and with minimal code, quickly turning projects into shareable web applications.
Streamlit is particularly popular for creating data dashboards, data visualization applications and Machine Learning model demos. *Streamlit*'s main strength is its simplicity. Creating web applications with just a few lines of Python code is possible. The programme abstracts from much of the complexity of web development, making it accessible even to those without extensive web development experience.
In addition, *Streamlit* provides several widgets (e.g. sliders, buttons, text inputs) that can be used to build interactive user interfaces. These widgets allow users to interact with the application and provide input.
In addition, *Streamlit* simplifies many aspects of web application development and allows the application's appearance and behaviour to be customized by writing Python code.

Given the many potentialities of this framework, it is decided to use it for the development of the user interface in order to develop a Web App that can interact with the user and assist them in recommending recipes.

5.2 Data storage: Firebase

Firebase is a powerful online service for storing and synchronising data processed by web and mobile applications. It is a resourceful NoSQL database with high availability and can be integrated quickly into other software projects simply by subscribing to the service.
Firebase belongs to a category of very rapidly spreading online services known as back-ends, the need for which has become increasingly evident in recent years. The growing capabilities of mobile devices and operating systems, together with increasingly widespread and fast network connections, have offered new tools to the imagination of developers. In this context, new applications tend to be less and less "locked" in devices and more capable of interacting with remote services, creating users social networks.
For all this to work, a server counterpart is needed that can make available, via APIs, services such as authentication, data storage, push notifications, communication between users and much more: the back-end.
Creating a back-end on one's own can lead developers to encounter difficulties of various kinds. First of all, of a technical nature, programmers may need more server skills and

experience to create a quality, secure and efficient back-end.
Furthermore, there may be organizational difficulties, for example, in reconciling the time and human resources involved in developing the server and client components.
Finally, there are infrastructural problems. People live in the age of the Cloud, of resources and functionalities offered as a service, in which, without having one's own physical or virtual machines, one can find network services that offer processing and storage in a reliable, flexible and cost-effective manner.
In cases where one requires a back-end for one's mobile or Web App but wants to avoid difficulties such as those just mentioned, one can rely on an online service offered in the Cloud and ready to interact with our software: Firebase is one such service.

In October 2014, a statement appeared on the project's official website, signed by James Tamplin, CEO and co-founder, announcing that Firebase was becoming part of Google's services [4]. This allowed Firebase to grow beyond its possibilities and the Mountain View company to include it in the Google Cloud Platform.
Tamplin stated in the post that in just three years, his crazy idea that could have worked had turned into a reality on which 110 000 developers based their work.
The success of Firebase is indisputably linked to its unique features:

- Ability to synchronize data and storage: Firebase can update data instantaneously, whether integrated with web or mobile apps. As soon as the app regains connectivity, it synchronizes the data displayed with the latest changes;
- Availability of client libraries to integrate Firebase into any app;
- REST APIs: make Firebase functionality available for any technology for which there are no dedicated libraries or for operations not covered by them;
- Security: The data stored in Firebase is replicated and backed up continuously. Communication with clients is always encrypted via SSL with 2048-bit certificates.

It can, therefore, be said that Firebase is a serverless platform for developing mobile and web applications. Open source but supported by Google, Firebase leverages Google's infrastructure and its Cloud to provide a suite of tools for writing, analysing and maintaining cross-platform applications. In fact, Firebase offers functionalities such as analysis, databases (using NoSQL structures), messaging and crash reporting for managing web, iOS and Android applications.

In particular, some of Firebase's primary services are exploited to develop the user interface, namely the authentication mechanism, to implement user authentication in applications using e-mail/passwords easily, social media logins (e.g. Google, Facebook, Twitter) and more, and real-time database. Indeed, Firebase provides a NoSQL cloud database that allows storing and synchronising data in real time across multiple clients, which is particularly useful for creating collaborative and interactive applications.

5.3 The Recipe Recommendation System Web App

This section will describe the Web App pages developed with *Streamlit* so that the user can interact with the generated recipe Recommendation System.

5.3.1 Welcome Page

It starts with a Welcome page, shown in Figure 5.1, which is often called the landing page or homepage and is the initial Web page that users encounter when they first visit the Web App. It is the web app's virtual "front door" and describes its main purpose.

Figure 5.1: Web App Welcome Page

5.3.2 Login Page

At this point, the user is presented with a Login page, Figure 5.2, which is specifically designed to capture the user's authentication credentials, namely the e-mail and password. It protects user accounts and data while providing registered users with a secure and easy access point. It plays a crucial role in ensuring access to the Web App, allowing registered users to log in and access their accounts.
In addition, a forgotten password link has been included to help users recover their accounts if they need help remembering their passwords and a registration link for new users who still need an account.
An error-handling mechanism is also developed to notify users of authentication errors, such as incorrect credentials.

In this case, Firebase Authentication is used to simplify the process of implementing user authentication, which provides a secure and scalable solution and allows developers to concentrate on implementing the core functionality of their applications, taking care of user management and security.
Indeed, when users interact with an application where this service has been adopted, they are asked to log in. The user must provide authentication information at this point, and Firebase Authentication validates the credentials. If these are found to be correct, it issues an authentication token. The application uses this token to identify the user and grant access to protected resources or functionality.

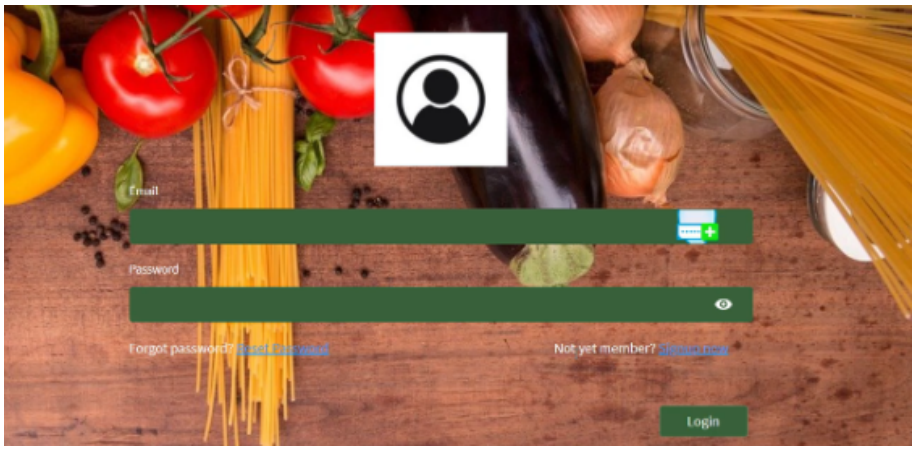

Figure 5.2: Web App Login Page

5.3.3 Password Reset Page

If the user does not remember his password, he can retrieve his account from a link on the Login page. Firebase Authentication again manages this recovery mechanism.

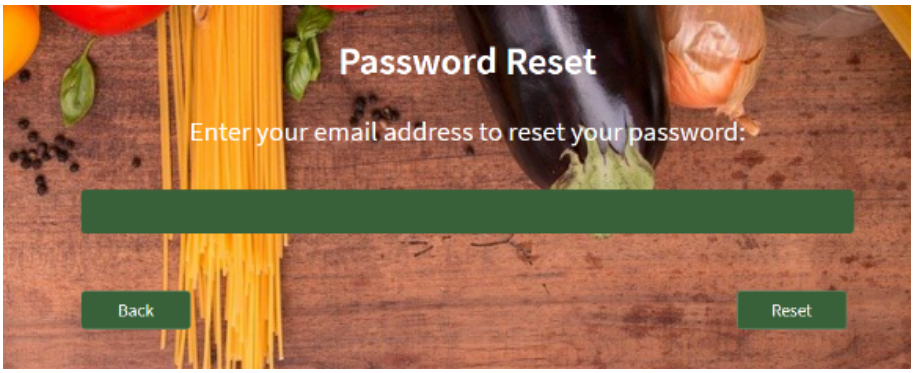

Figure 5.3: Web App Password Reset Page

5.3.4 User Registration Page

New users who do not yet have an account can, instead, access via a link to the Registration page, shown in Figure 5.4, where they will be asked for their first name, surname, e-mail, password, confirmation of the latter, gender, date of birth, weight, height, any food allergies and whether they are vegetarian or vegan. This allows users to create an account and access the Web App's features.
Firebase is once again used to handle user input from the registration form: the *createUserWithEmailAndPassword* method of Firebase Authentication is used to create a new user account with the provided e-mail and password, checking whether the provided e-mail is valid or already exists and whether the password meets the security criteria. In this way, it is possible to securely store and manage users' e-mail addresses and passwords while guaranteeing data security and authentication; all other inputs, however, are stored with the help of Firebase Realtime Database, which, unlike Firebase Authentication, is designed to store structured user data.

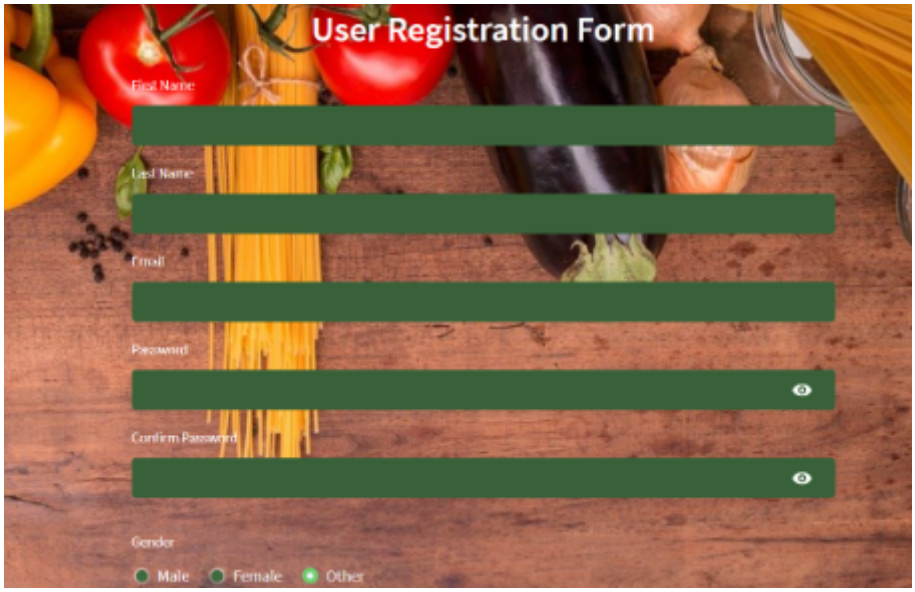

Figure 5.4: Web App User Registration Page

5.3.5 Recipes Choice Page

After completing the registration form, based on the information obtained via the Firebase Realtime Database, an algorithm is applied at the back-end to filter, among the recipes present in FoodKG, those that can satisfy the users' dietary constraints. A sub-graph of candidate recipes is then extracted from FoodKG, from which ten are selected to be proposed to the user, and a link is created for them with Recipe1M in such a way as to show the user not only the title of the recipes but also their images.
In order to incorporate the user's preferences in the recipe Recommendation System, the user will be shown a new Web page, shown in Figure 5.5, in which ten recipes are proposed to them, and they are asked to select five of them.
This step is of fundamental importance as it allows the user's first interactions with the recipe Recommendation System to be collected: the recipes selected are stored in the Firebase Realtime Database as preferences, while those not selected constitute non-preferences. This ends the registration process, and, if successful, the user is asked to authenticate themself via the Login page.

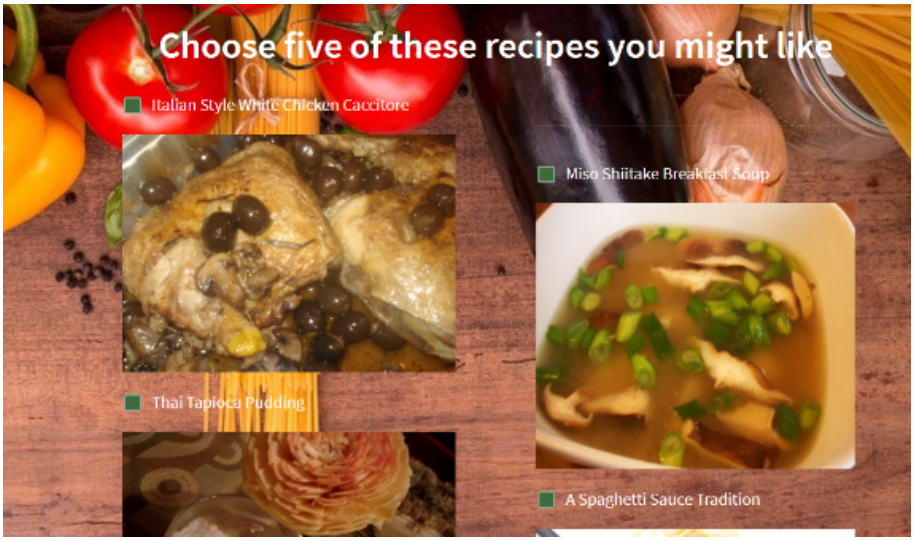

Figure 5.5: Web App Recipe Choice Page

5.3.6 Welcome - Back Page

Once authenticated, the user is presented with a new Home page, shown in Figure 5.6, where they can choose whether they want to access their account information or be guided in choosing a recipe.

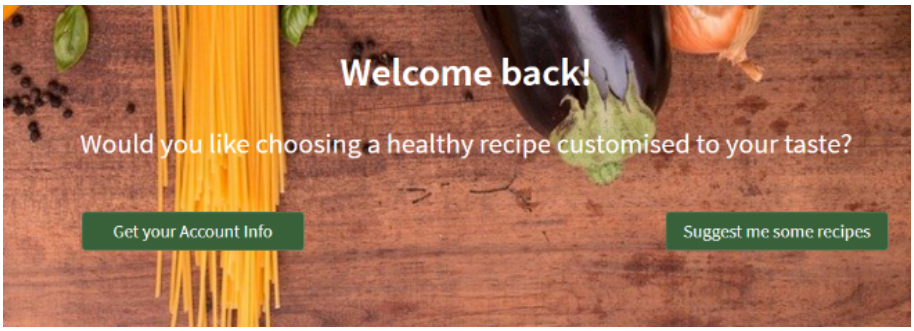

Figure 5.6: Web App Welcome - Back Page

5.3.7 User Profile Page

Suppose the users wish to view their account information. In that case, they are given access to the User Profile page where, through the use of Firebase Realtime Database, the user can view the information they have entered during registration or previously modified. Again, using Firebase Realtime Database, users can update their profile information.
The user can delete their account or log out using Firebase Authentication permanently via the page shown in Figure 5.7.

Figure 5.7: Web App User Profile Page

5.3.8 Delete Account Page

If the users have decided to remove their account permanently, they are shown the Web page in Figure 5.8, where they are asked whether they are sure of their choice.

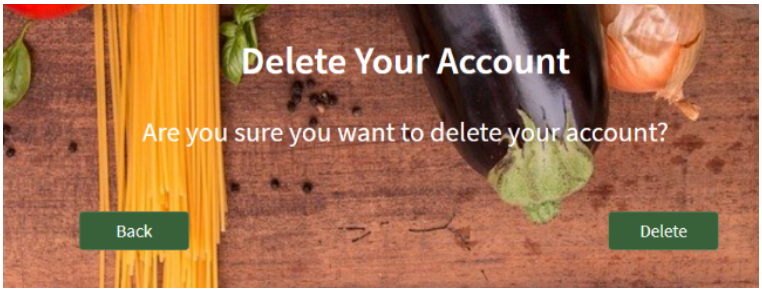

Figure 5.8: Web App Delete Account Page

5.3.9 Logout Account Page

If, on the other hand, the users have decided to log out of their account, they are presented with the page shown in Figure 5.9, where they are asked whether they are sure of their choice.

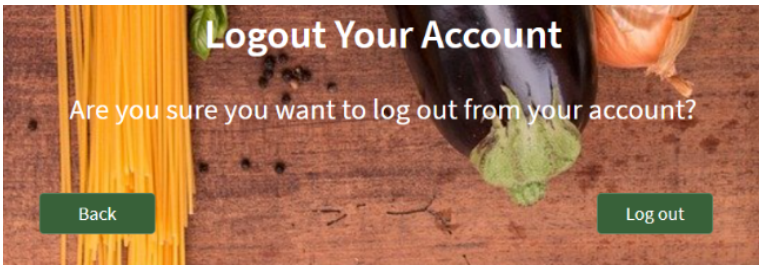

Figure 5.9: Web App Logout Account Page

5.3.10 Recommended Recipes Choice Page

If, after authenticating themselves, the user decides to be guided in the recommendation of specific recipes, the implemented Recommendation System is used in the back-end to generate the six recipes best suited to their preferences.
Also, in the back-end, a link is created with the Recipe1M dataset to obtain the associated images from the title of the recipes. At this point, the page shown in Figure 5.10 is generated and presented to the user, who is prompted to select the preferred recipe. This recipe is added to the user's preferences.

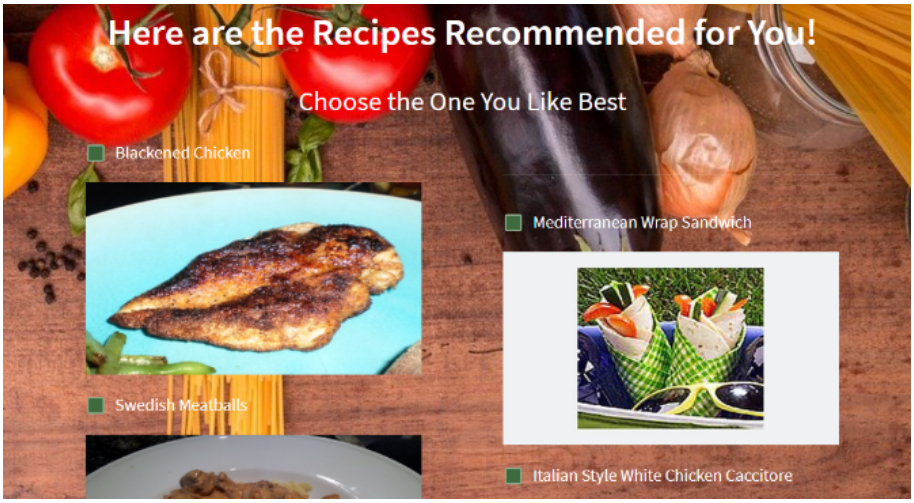

Figure 5.10: Web App Recommended Recipes Choice Page

5.3.11 Recipe Instruction Page

Starting from the selected recipe, the Recipe1M dataset is once again used to obtain not only the image of the recipe but also information on its ingredients and instructions. In this way, the user is shown a new page in Figure 5.11, in which the user is guided step by step in realising the chosen recipe.

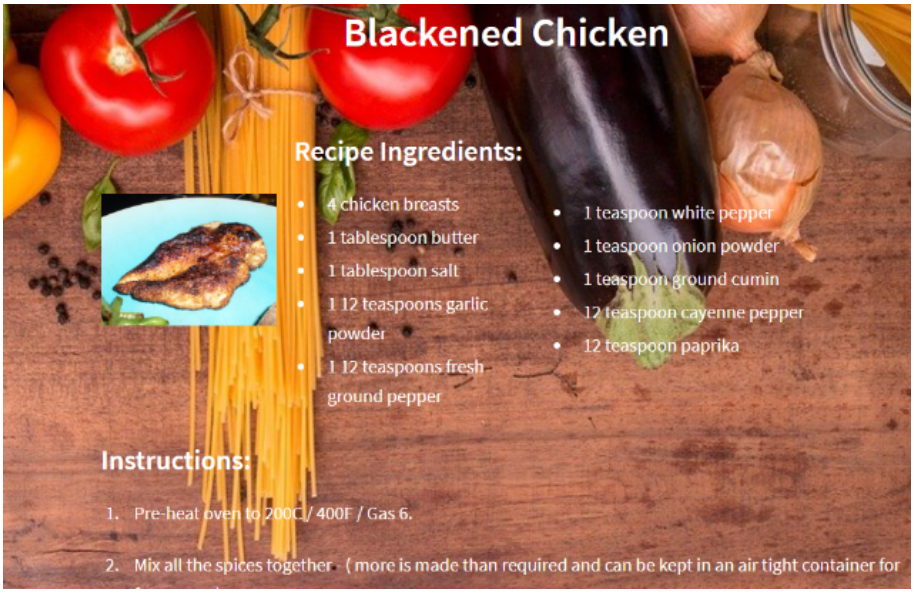

Figure 5.11: Web App Recipe Instruction Page

5.3.12 Navigation Diagram

Subsequently, starting from the pages constituting the Web App, the navigation diagram shown in Figure 5.12 is constructed, which represents the sequence of screens that can be viewed while using the Web App.

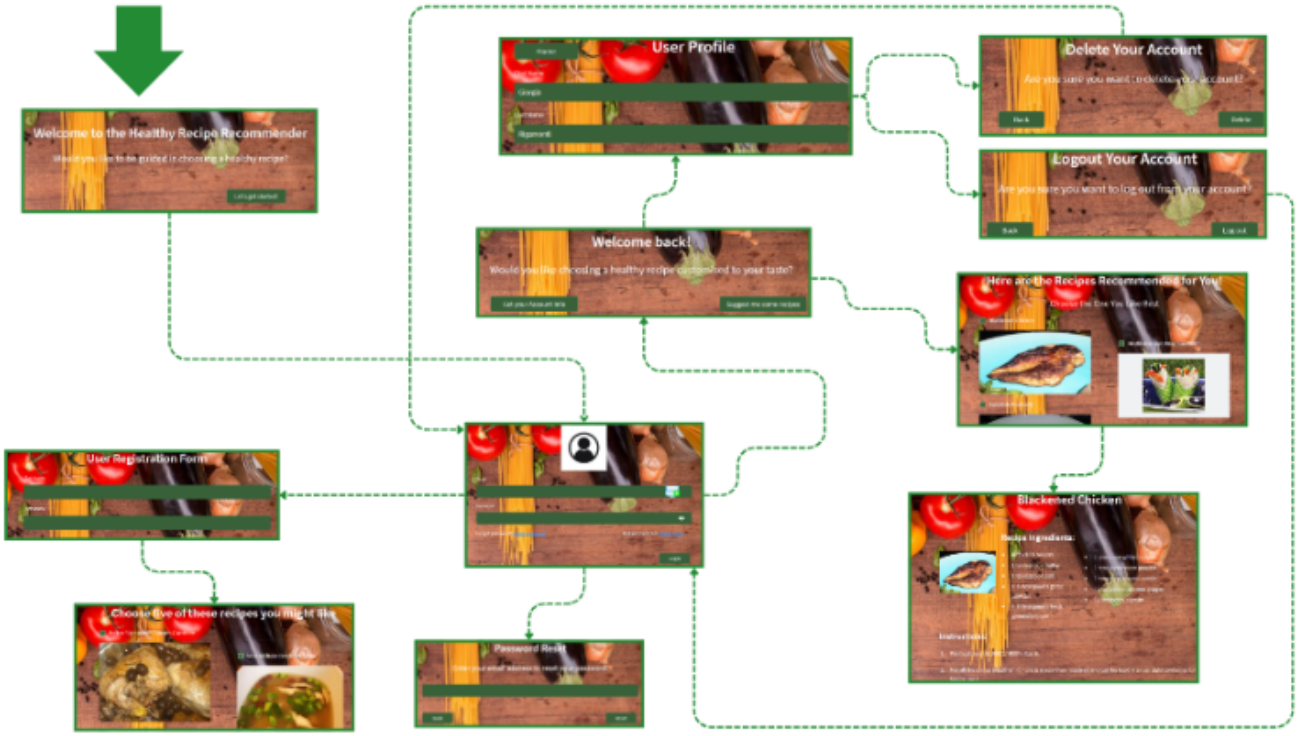

Figure 5.12: Web App Navigation Diagram

5.4 Two examples of the Web App usage

This section will present two examples of Web App use by two different users.

5.4.1 The First Example

In this example, suppose a new user logs into the Web App and wishes to be supported in choosing a new recipe.
Then, after viewing the Welcome page, he is shown the Login page, and since he still needs to register, he clicks on the appropriate link and begins his registration, as shown in Figure 5.13.

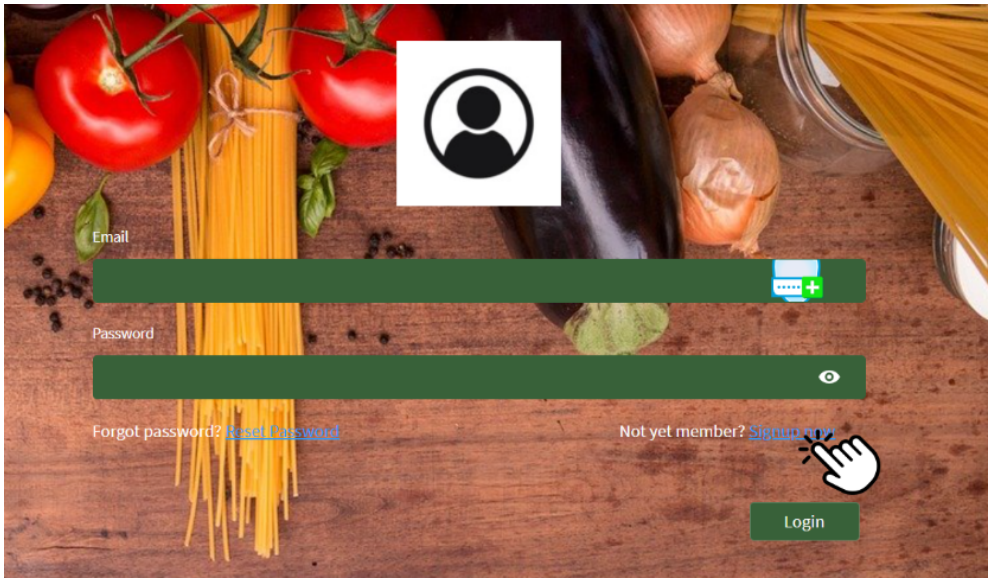

Figure 5.13: Login Page and click on the SignUp Now link

Specifically, as depicted in Figure 5.14, the user in question is Federica Negri, and she was born on 18/08/1995, is 156 cm tall, weighs 58 kg, has a sedentary lifestyle, and is allergic to milk and nuts.

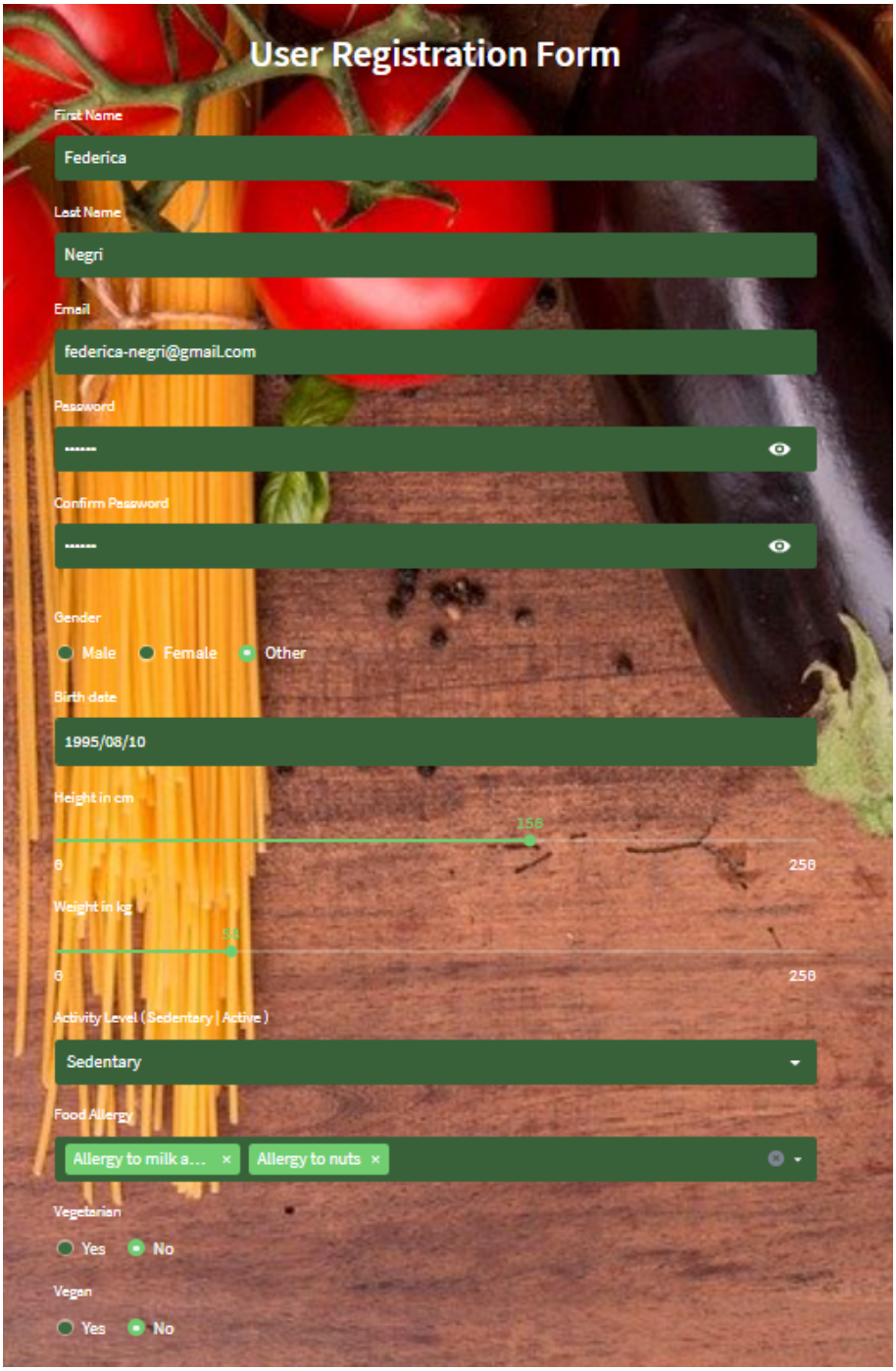

Figure 5.14: Federica Negri Registration Form

After filling in the Registration Form, based on the information entered by the user, a FoodKG sub-graph containing all the recipes suitable for the user is retrieved in the back-end, and ten of them are extracted. At this point, the Web App shows the user these recipes, and he is invited to select five of them that he prefers.

In this case, as reported in Figure 5.15, the user Federica is shown the following recipes:

- Simmered Chicken Wings and Daikon (White Radish)
- Braised Chicken with Garlic and White Wine
- Vietnamese Shrimp Spring Rolls
- Terodactyl Wings
- Cornish Game Hens with Pancetta, Juniper Berries and Beets
- Diet salad
- Tomato Matzo Balls
- Hong Kong Style Spicy Deep Fried Shrimp
- Adobo Chicken with Mango and Avocado
- Delicious Low-Fat Ginger Molasses Cookies (Healthy!)

Figure 5.15: Choosing the five recipes

Of whom, it chooses:

- Braised Chicken with Garlic and White Wine
- Vietnamese Shrimp Spring Rolls
- Diet salad
- Adobo Chicken with Mango and Avocado
- Delicious Low-Fat Ginger Molasses Cookies (Healthy!)

In this way, she has completed her registration and when she wants to have a recipe recommended according to her tastes and allergies, she can do so directly after login.

Assume now, that she wants a recipe recommended especially for her: she logs into the Web App and clicks on the "Suggest me some recipes" button, as displayed in Figure 5.16.

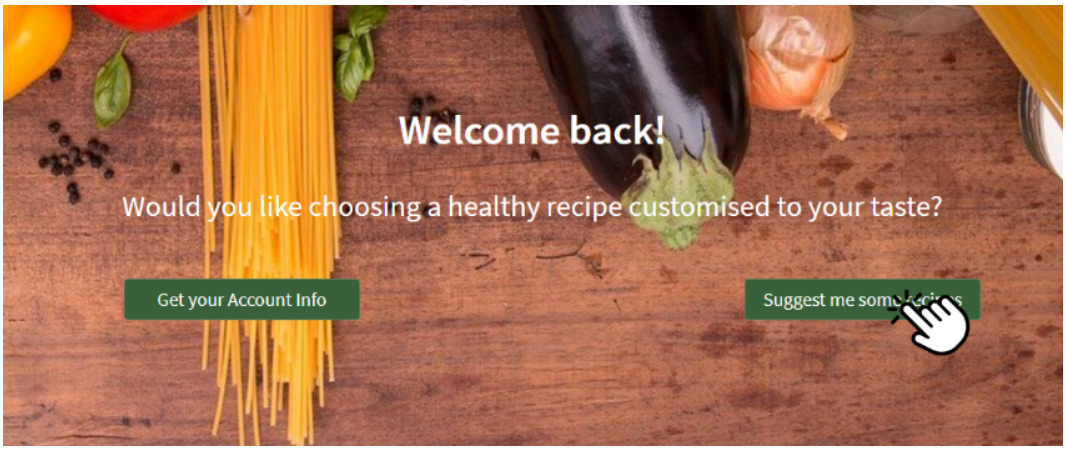

Figure 5.16: Welcome Back Page and suggest recipes click

The Recommendation System, in the back-end, considers all the recipes she has preferred and goes on to recommend the six recipes that come closest to her tastes and displays them via the Web App. These recipes are shown in Figure 5.17:

- Mango Salsa

- Soba Noodles in Cool Lime & Soy Sauce Broth with Chicken & Fresh Asian Turnip
- COLESLAW
- Cajun Roast Potatoes
- Crispy Chinese Noodles with Eggplant and Peanuts
- San Franciscan Cioppino Stew

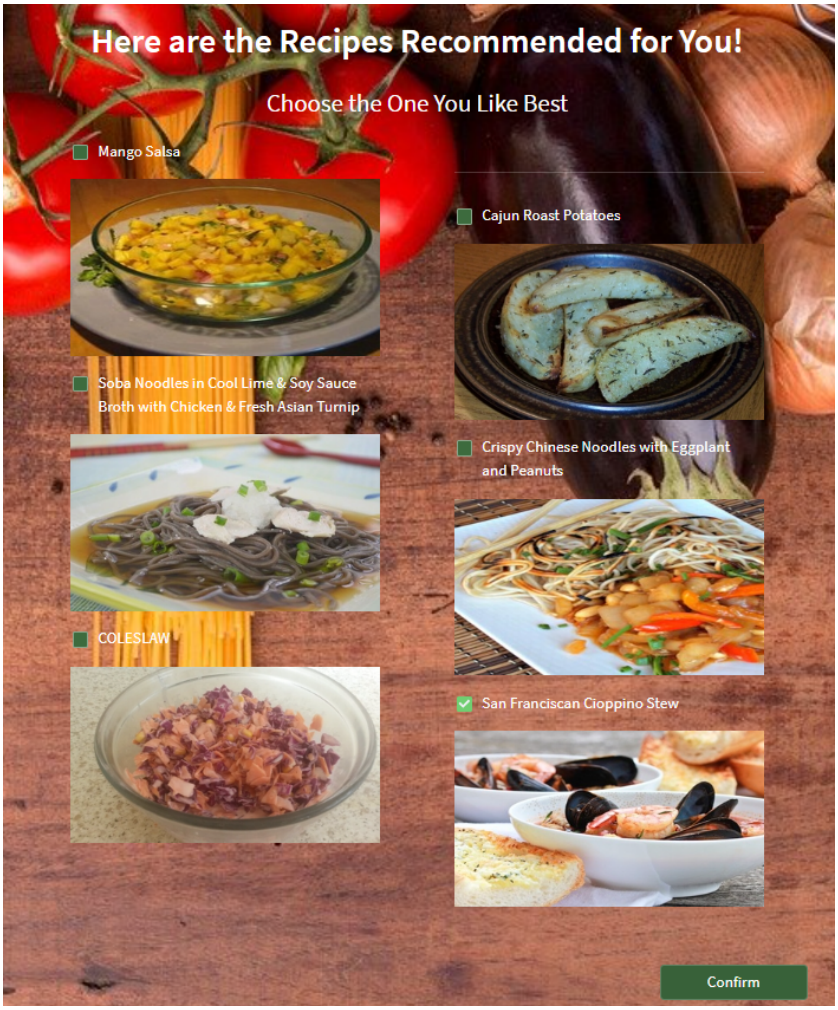

Figure 5.17: Federica's Recommended Recipes Page

Assume that among the six, she has a greater preference for the recipe "San Franciscan Cioppino Stew".
At this point, as can be seen in Figure 5.18, the Web App shows Federica all the necessary ingredients and basic steps to make the recipe.

Figure 5.18: The Selected Recipe Instruction Page

5.4.2 The Second Example

Similarly to the example above, consider now a new user who accesses the Recommendation System Web App for the first time. He proceeds in the same way as in the previous example and displays the Registration page.
In this case, as can be viewed in Figure 5.19, the user in question is Mario Rossi, born on 18/06/1990, is 182 cm tall, weighs 75 kg, has an active lifestyle, and is allergic to eggs and vegetarian.

Figure 5.19: Mario Rossi Registration Form

He also fills in the Registration Form, and after a FoodKG subgraph containing all the recipes suitable for him is extracted at the back-end, he displays the following recipes (as also shown in Figure 5.20) via the Web App:

- Brandied Orange and Cranberry Sauce
- Angel Biscuits II
- Mediterranean Roasted Chickpea Salad
- Saturday Sourdough Bread
- Aubergine (Eggplant) Curry (2)
- Copycat Pace Picante Sauce
- Herb Roasted Almonds
- Kidney Bean, Red Onion And Tomato Salad
- Sage and Garlic Roasted Potatoes
- MIRACLE WHIP Make-ahead Buffet Salad

Figure 5.20: Choosing the five recipes

Of these, he chooses the following five:

- Angel Biscuits II
- Mediterranean Roasted Chickpea Salad
- Aubergine (Eggplant) Curry (2)
- Kidney Bean, Red Onion And Tomato Salad
- MIRACLE WHIP Make-ahead Buffet Salad

Having completed the registration phase, Mario decides to login to be supported in the choice of a recipe; similarly to the previous case, after login into the Welcome Back page, he presses the button "Suggest me some recipes" and the Web App proceeds to expose to him the six recipes most suitable for him, as indicated in Figure 5.21:

- Grilled Eggplant (Aubergine) Parmesan
- Avocado Tzatziki
- Spiced Coffee
- Banana Split Dessert
- Orange Fennel Salad
- Black Bean Enchilada Quinoa

Figure 5.21: Mario's Recommended Recipes Page

Of these six, he prefers the "Black Bean Enchilada Quinoa" recipe and then selects it. Finally, as illustrated in Figure 5.22, the Web App shows him the steps to make it.

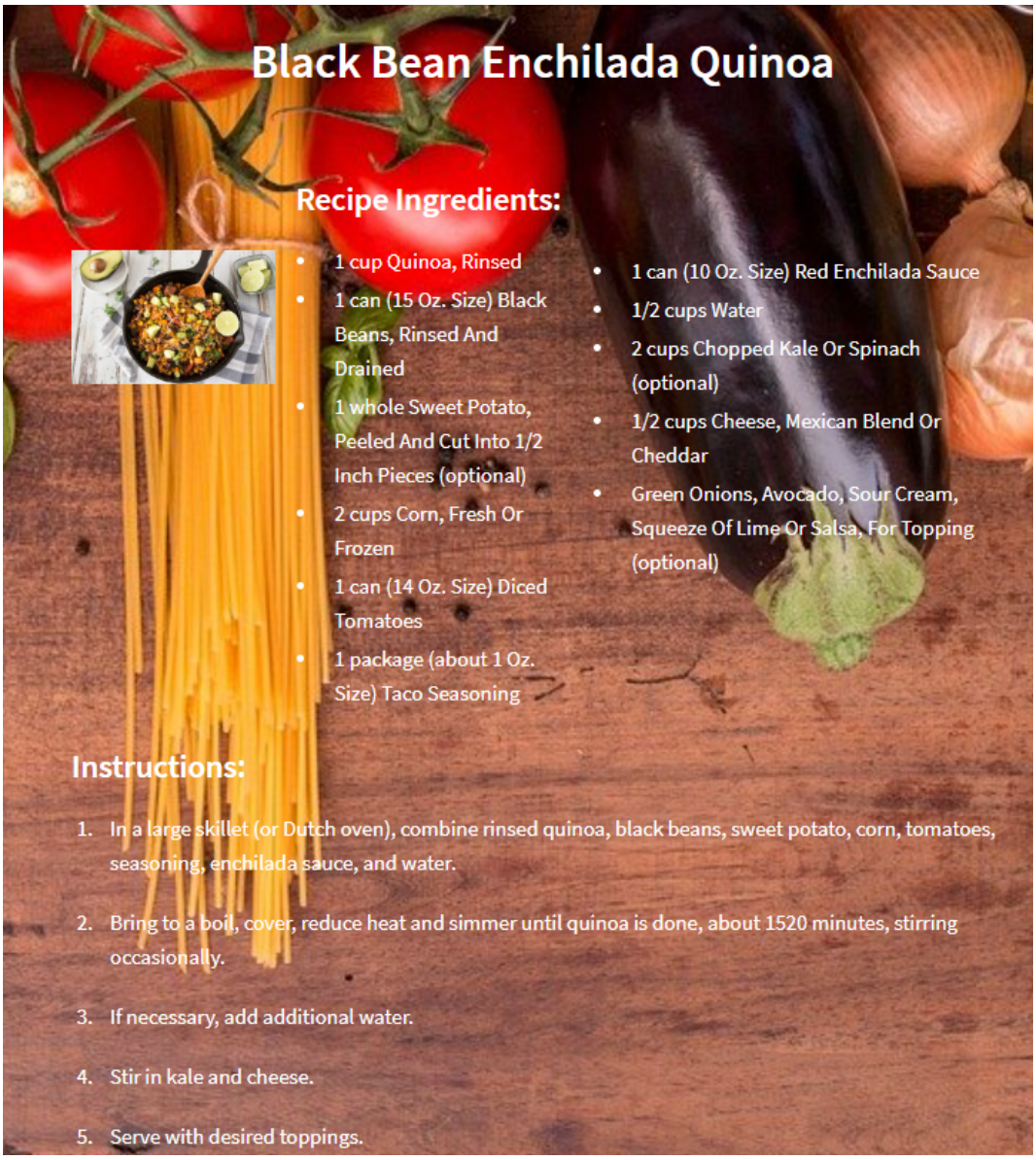

Figure 5.22: The Selected Recipe Instruction Page

Chapter 6

Conclusions and Future Developments

Currently, Recommendation Systems play a fundamental part in different realities. In a world where social media is growing increasingly, and the user is inundated with thousands of data, these recommendation techniques give an essential advantage to the system in which they are implemented. Recommendation tools play a significant role on various platforms. Their inclusion in systems makes it possible to create more user-oriented applications.
The importance of Recommendation Systems lies in allowing a personalized and close-up view of the end user.
Recommendation Systems have been used to solve the problem of information overload by suggesting related and relevant elements to users. These filtering techniques learn data from user behaviour and exploit this information to recommend something they like, and that matches their preferences, likes and interests.
The objective of this thesis was to apply Artificial Intelligence techniques, in particular recommendation techniques, to the development of a content-based Recommendation System of personalized recipes oriented towards the user's health, obtained with the aid of FoodKG, an extensive knowledge graph aimed at collecting the semantic relationships between recipes and their attributes (recipe titles, ingredients, quantities, units, ...).
In order to achieve this objective, it was necessary to start with an in-depth study of the Recommendation Systems that are currently adopted in various reference domains, with a particular focus on an area in which there is still much work to be done: Recommendation Systems in the food sphere, which appear to play a fundamental role in guiding the user's choices and thus also influencing their behaviour.

Indeed, Food Recommendation Systems not only provide relevant recommendations that users may wish to eat but can also help users to consume a healthier diet. More specifically, from this analysis, recent interest has emerged in developing Recommendation Systems based on knowledge graphs that enable a Recommendation System better to understand the latent reciprocal relationships between different items and consequently make the results of recommendations more explainable.
Starting from this, it was necessary to reconstruct the FoodKG knowledge graph in order to achieve the set objective, which was then extended in such a way that it could also include labels concerning the wholesomeness of recipes and the possible presence of allergens or the unsuitability of ingredients for particular food diets.
At this point, it was continued with the study of different types of graph neural networks to identify the one most suitable for constructing the Recommendation System based on FoodKG: GraphSage.
Thanks to the latter, in fact, it was possible to develop and implement the Recommendation System, and, despite the few initial user interactions on the FoodKG recipes, it was also possible to obtain discrete performances on the model, which later led to its use for the generation of recommendations. At this point, it was necessary to implement a Web App that made it possible for the newly created Recommendation System to interface with the user.

It is, therefore, necessary to recognize the potential of the Recommendation System created, considering that its performance could significantly increase as the number of users and interactions between these users and the FoodKG recipes increases. Indeed, in the future, it will be necessary to further extend FoodKG in such a way that it also includes a classification of recipes concerning the reference meal and also to consider the substitution of certain ingredients in the recipes in order to make the recipe in question suitable for people with a particular allergy, who are vegetarians or vegans or who have a specific pathology that is also influenced by diet (e.g. diabetic patients).
In addition, the current Recommendation System could be further improved to recommend recipes considering, in addition to the constraints already introduced previously, only the ingredients available to the user so that it could adapt more to different situations and be increasingly customized.
What is more, as time passes in which the Recipe Recommendation System is used, there will be an increasing number of user interactions, and the number of users will also increase. Consequently, one could also consider periodically upgrading the GraphSAGE model with the newly collected data and, at the same time, improve the quality of the recommendations.

In this regard, once a considerable amount of user interaction data has been collected, it might be interesting to explore advanced techniques such as collaborative filtering and to convert the content-based Recommendation System into a hybrid system that can combine content-based filters with collaborative filters. Indeed, creating a hybrid Recommendation System will allow the strengths of both collaborative and content-based filtering to be exploited while mitigating its weaknesses. This may lead to more accurate and diverse recommendations, ultimately improving the user experience.

References

[1] Blazegraph database. https://blazegraph.com/.

[2] A comprehensive case-study of graphsage. https://towardsdatascience.com/a-comprehensive-case-study-of-graphsage-algorithm-with-hands-on-experience-using-p

[3] Cosine similarity. https://www.geeksforgeeks.org/cosine-similarity/.

[4] Firebase. https://www.html.it/articoli/firebase/.

[5] Food and nutrient database for dietary studies (fndds). https://catalog.data.gov/dataset/food-and-nutrient-database-for-dietary-studies-fndds-82071.

[6] Keras. https://keras.io/.

[7] Min max scaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[8] Nltk. https://www.nltk.org/.

[9] Recommendation systems. https://towardsdatascience.com/addressing-the-recommendation-task-through-a-different-lens-3226998bad58.

[10] Recommendation systems based on kg. https://towardsdatascience.com/introduction-to-knowledge-graph-based-recommender-systems-34254efd1960.

[11] Streamlit. https://streamlit.io/.

[12] Tf-idf. https://www.okpedia.it/tf-idf.

[13] Word2vec. https://towardsdatascience.com/word2vec-explained-49c52b4ccb71.

[14] *Principal Component Analysis for Special Types of Data*, pages 338–372. Springer New York, New York, NY, 2002.

[15] Foodon: A global farm-to-fork food ontology - the development of a universal food vocabulary. In *2016 Joint International Conference on Biological Ontology and BioCreative - Food, Nutrition, Health and Environment for the 9 Billion, ICBO-BioCreative 2016*. CEUR-WS, January 2016.

[16] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[17] Wu Y. Chen, H. Risk assessment of food allergens. *China CDC weekly*, 2022.

[18] Damion Dooley, Emma Griffiths, Gurinder Gosal, Pier Luigi Buttigieg, Robert Hoehndorf, Matthew Lange, Lynn Schriml, Fiona Brinkman, and William Hsiao. Foodon: a harmonized food ontology to increase global food traceability, quality control and data integration. *npj Science of Food*, 2, 12 2018.

[19] Mouzhi Ge, Francesco Ricci, and David Massimo. Health-aware food recommender system. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, page 333–334, New York, NY, USA, 2015. Association for Computing Machinery.

[20] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 1970.

[21] Paul Groth, Andrew Gibson, and Jan Velterop. The anatomy of a nanopublication. *Inf. Serv. Use*, 30:51–56, 2010.

[22] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[23] Steven Haussmann, Oshani Seneviratne, Yu Chen, Yarden Ne'eman, James Codella, Ching-Hua Chen, Deborah Mcguinness, and Mohammed Zaki. *FoodKG: A Semantics-Driven Knowledge Graph for Food Recommendation*, pages 146–162. 10 2019.

[24] Jon Herlocker, Joseph Konstan, and John Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information Retrieval*, 5:287–310, 01 2002.

[25] L. Hirschman and R. Gaiazauskas. Natural language question answering: the view from here. *Natural Language Engineering*, 7(4):275–300, 2001.

[26] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE international conference on data mining*, pages 263–272. Ieee, 2008.

[27] Shyong Lam, Dan Frankowski, and John Riedl. Do you trust your recommendations? an exploration of security and privacy issues in recommender systems. pages 14–29, 01 2006.

[28] Diya Li, Mohammed J. Zaki, and Ching hua Chen. Health-guided recipe recommendation over knowledge graphs. *Journal of Web Semantics*, 75:100743, 2023.

[29] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4, Part 2):2065–2073, 2014.

[30] Javier Marin, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019.

[31] Levin S. Melina V, Craig W. Position of the academy of nutrition and dietetics: Vegetarian diets. *Vegetarian Diets. J Acad Nutr Diet.*, 2016.

[32] Weiqing Min, Chunlin Liu, Leyi Xu, and Shuqiang Jiang. Applications of knowledge graphs for food science and industry. *Patterns*, 3(5):100484, 2022.

[33] Chizuru Nishida, Ricardo Uauy, Shiriki Kumanyika, and PS Shetty. The joint who/fao expert consultation on diet, nutrition and the prevention of chronic diseases: Process, product and policy implications. *Public health nutrition*, 7:245–50, 03 2004.

[34] Sabbir Rashid, Katherine Chastain, Jeanette Stingone, Deborah Mcguinness, and Jamie McCusker. The semantic data dictionary approach to data annotation integration. 10 2017.

[35] Gary Sacks, Mike Rayner, and Boyd Swinburn. Impact of front-of-pack 'traffic-light' nutrition labelling on consumer food purchases in the uk. *Health promotion international*, 24:344–52, 10 2009.

[36] Karypis G. Konstan J. Sarwar, B. and J. Riedl. Item-based collaborative filtering recommendation algorithms. *In Proceedings of the 10th International Conference on World Wide Web (New York, NY, USA, 2001), WWW*, pages 285—295, 2001.

[37] Christoph Trattner and David Elsweiler. Investigating the healthiness of internet-sourced recipes: Implications for meal planning and recommender systems. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 489–498, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[38] Fan W. Epidemiology in diabetes mellitus and cardiovascular disease. *Cardiovasc Endocrinol.*, 2017.

[39] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. pages 417–426, 10 2018.

[40] Zuoshuang Xiang, Mélanie Courtot, Ryan Brinkman, Alan Ruttenberg, and Yongqun He. Ontofox: Web-based support for ontology reuse. *BMC research notes*, 3:175, 06 2010.

[41] Longqi Yang, Andy Hsieh, Hongjian Yang, Nicola Dell, Serge Belongie, and Deborah Estrin. Yum-me: Personalized healthy meal recommender system. 05 2016.

[42] Kam Fung Yeung and Yanyan Yang. A proactive personalized mobile news recommendation system. In *2010 Developments in E-systems Engineering*, pages 207–212, 2010.

[43] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. Personalized entity recommendation: A heterogeneous information network approach. In *WSDM 2014 - Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM 2014 - Proceedings of the 7th ACM International Conference on Web Search and Data Mining, pages 283–292. Association for Computing Machinery, 2014.

[44] Fuzheng Zhang, Nicholas Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. pages 353–362, 08 2016.

[45] Yongfeng Zhang, Qingyao Ai, Xu Chen, and Pengfei Wang. Learning over knowledge-base embeddings for recommendation. 03 2018.

[46] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lee. Meta-graph based recommendation fusion over heterogeneous information networks. 08 2017.